CONVERGYS

Intelligent Voice Portal

version 6.7

Interaction Composer Cookbook

60001693
September 2013

Trademarks

Convergys refers to Convergys Corporation or any of its wholly owned subsidiaries.

Outthinking. Outdoing. is a registered service mark of Convergys. Convergys, the Convergys logo, Edify Voice Interaction Platform, Edify, Electronic Workforce, EVIP, Intervoice, InVision, Media Exchange and RealCare are trademarks and/or registered trademarks of Convergys in the United States and/or other countries, including but not limited to in Chile, China, Costa Rica, Guatemala, and Honduras, where Convergys is a registered trademark. Parts of the described systems may be covered by one or more issued or pending U.S. or other patents.

All other trademarks not owned by Convergys that appear in this document are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Convergys or its subsidiaries.

If Customer is a U.S. Government user, the System, Software, Functional Specifications and/or Documentation are provided with RESTRICTED RIGHTS. Use, duplication and disclosure by the U.S. Government are subject to restrictions as set forth in Federal Acquisition Regulation (FAR) clause 52.227-19 (c)(2) or successor regulation thereto. Manufacturer is Convergys Corporation or its subsidiaries, 201 East Fourth Street, Cincinnati, Ohio 45202.

The products, specifications and content described in this document are subject to continuous development and improvement and Convergys is entitled to change them at any time. Convergys is not liable for loss or damage of any nature whatsoever arising or resulting from the use of or reliance on outdated information. Convergys does not warrant that the material contained in this documentation is free of errors. Any errors found in this document should be reported to Convergys in writing.

THIS SYSTEM, SOFTWARE, FUNCTIONAL SPECIFICATIONS AND/OR DOCUMENTATION CONTAIN CONFIDENTIAL INFORMATION AND TRADE SECRETS OF CONVERGYS. USE, DISCLOSURE, AND REPRODUCTION ARE PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF AN OFFICER OF CONVERGYS.

Any entity or person with access to this information shall be subject to this confidentiality statement. If you have received this document via e-mail in error, please send the e-mail to the originator, indicating that you received it in error. If you have received a hard copy of this document in error, mail this document in its entirety to Convergys Corporation, 201 East Fourth Street, Cincinnati, Ohio USA 45202.

Windows NT, Windows 2000, Windows 2003, Windows 7, and Windows 2008 are registered trademarks of Microsoft.

This product may include software developed under Apache Software License, Version 2. Copyright © 2004-2013 The Apache Software Foundation (http://www.apache.org/). All rights reserved.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Table of Contents

Table of Figures

An Introduction to Interaction Composer Studio

Overview

Interaction Composer Studio (IC or ICS) is Convergys' strategic graphical tool for developing interactive voice applications. IC uses a graphical flowchart paradigm which turns dialog design into an easier process of drawing a flowchart. IC's architecture also provides support for future extensions that enable multimodal application design.

The IC flowchart tool hides many of the complex details involved in building a voice application, allowing an application developer to focus on the high-level dialog flow. Much of the complexity of an automated dialog deals with user-entry timeouts, error recovery, reprompting schedules and other issues. IC defaults these issues to industry-standard rules and values, allowing the developer to concentrate on the main dialog flow and user experience. The developer can still access these dialog details in IC, but IC keeps the details out of the way until needed.

Typically, the application developer should use IC initially to define the high-level dialog flow, without worrying about the details of error recovery and validation. The goal of the dialog designer is to make the interactive dialogs as natural as possible, allowing callers to interact with their information with minimal effort and frustration. IC lets the developer focus on those issues. Once the main flow has been established, then the designer can work on lower-level issues such as error recovery and database interfaces.

Why Should I Read this Cookbook?

Each cookbook application is essentially a "recipe" for a common type of application.

Read this cookbook:

- To learn how to use Interaction Composer to build interactive voice dialogs quickly and efficiently.
- To see how Interaction Composer works before committing to learning the language and/or to building a full application. Many developers want to learn a new tool by writing a basic application, without having to reference all the features and options.
- To start developing with Interaction Composer as soon as possible. Pick two or three recipes that are similar to the kinds of applications you intend to build. Later, you can combine these pieces into your application.
- To get an overview of the features and functionality of Interaction Composer by following a few of the step-by-step recipes for simple applications.
- To learn dialog design as a beginner. If you have never designed a voice dialog, the cookbook will lead you through the process, step-by-step and you can use the Interaction Composer tool to build:
    - A basic Hello World voice application
    - These applications cover the minimum steps and knowledge to create and run an application

This cookbook hits the highlights of IC without getting bogged down too much in the details of dialog design or the vast array of rarely-used options available with the IC components. One of the main purposes of Interaction Composer is to allow developers to design user-friendly dialogs without having to get into the details of VoiceXML, CCXML, or SCXML. IC defaults its' dialog error recovery and reprompting strategies to industry-standard methods, which provide reliable recovery processes if the developer doesn't want to deal with these issues.

The IC User's Manual is a reference manual. It provides details about each feature with all the options available. Typically, you would use it as you design an application and need to look up a specific feature of a component that you don't understand or to find a specific function.

While you read this cookbook, be sure to have Interaction Composer installed on your computer nearby. As you read through each recipe, you should follow along on your computer, creating the recipe.

Overview of Developing, Packaging, and Deploying

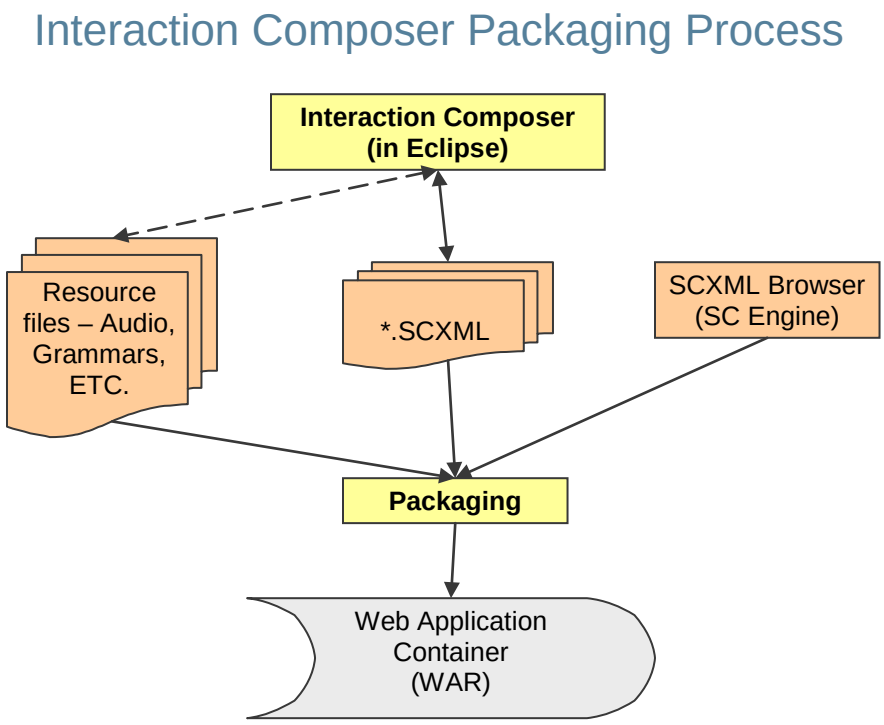
Here is an overview of how IC builds an application and how that application is deployed.

The application developer starts by launching Eclipse, which has been preconfigured with the IC plug-in.

- The Eclipse Integrated Development Environment (IDE) is a software framework that supports application software development.
- Eclipse is written primarily in Java for portability, but Eclipse doesn't promote a specific programming language. The Eclipse framework allows plugin components which can expose a specific programming methodology. Convergys has designed a plugin for Eclipse that exposes a graphical user interface to the voice application developer, and that UI is focused on allowing a developer to design voice dialogs by drawing flowchart diagrams.
- To make a developer's job easier, all Eclipse plugin components are bundled into the install program.

Then the developer creates a project in IC and draws a flowchart of the application flow in that project. As the developer designs the flow diagram, IC builds an internal data structure that captures the flow essentials. Each time the developer saves the flowchart, IC creates various files that contain all of the information defining the logic flow as well as the data required to display the flowchart icon positioning and connectivity.

When finished with the dialog design, the developer saves the project and then packages the application into a WAR (Web Application ARchive) file. The WAR file will contain all of the files and data necessary to run the application, including audio recordings, grammars, and SCXML documents, as well as an SCXML browser and any other runtime processes required to execute the application. This diagram illustrates the packaging process for a completed application and runtime execution.



To deploy the application, the developer loads the final WAR file into an application server. When the SCXML documents are executed, the code dynamically generates snippets of VoiceXML and CCXML code which are then presented to the appropriate client-side browsers to execute the dialog segments.

Recipe 1: Creating Your First IC Application "Hello World"

Approximate time: 30 minutes, depending on prior experience

Reference: For the *Interaction Composer User's Guide*, select **Help > Help Contents** and select this guide from the **Contents** pane.

What you will learn with this recipe:

- Creating an Application Project
- Creating a New Voice Diagram
- Creating a Call Diagram
- Playing a Prompt
- Using TTS for a Prompt
- Packaging Your Application Project

Creating an application project

To help you get familiar with Interaction Composer, the HelloWorld project will be very basic:

- Answers the phone
- Plays a "Hello World" prompt
- Hangs up

To start building your application project:

1. In Interaction Composer (IC), create a new workspace named **HelloWorld**. The Interaction Composer perspective opens in Eclipse.
2. When IC opens, close the **Getting Started** tab. You can read it another time.
3. From the IC menu bar, select **File > New > ICS Application Project**. See Figure 1.1.

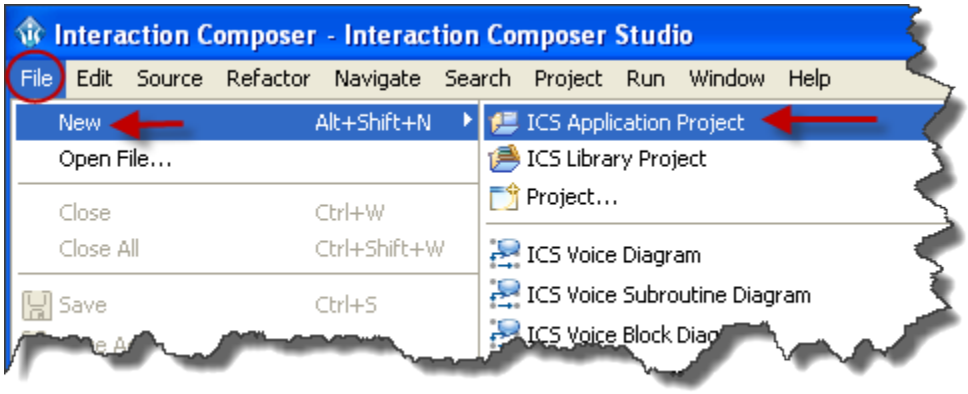


Figure 1.1- Creating a new application project

The New ICS Application Project wizard opens (see Figure 1.2).

4. In the Project name field, type the project name. For this recipe, type **HelloWorld**.

   Note: The project name has no limit on length, but it cannot have spaces. The name must be unique and cannot already exist in your workspace. The best practice is to name the project the same as your workspace. For example, the workspace HelloWorld would have the project name HelloWorld and the workspace Banking would have the project name Banking.

5. Typically, check the **Use default location** checkbox. This is checked by default. This will put the project into your default workspace folder.

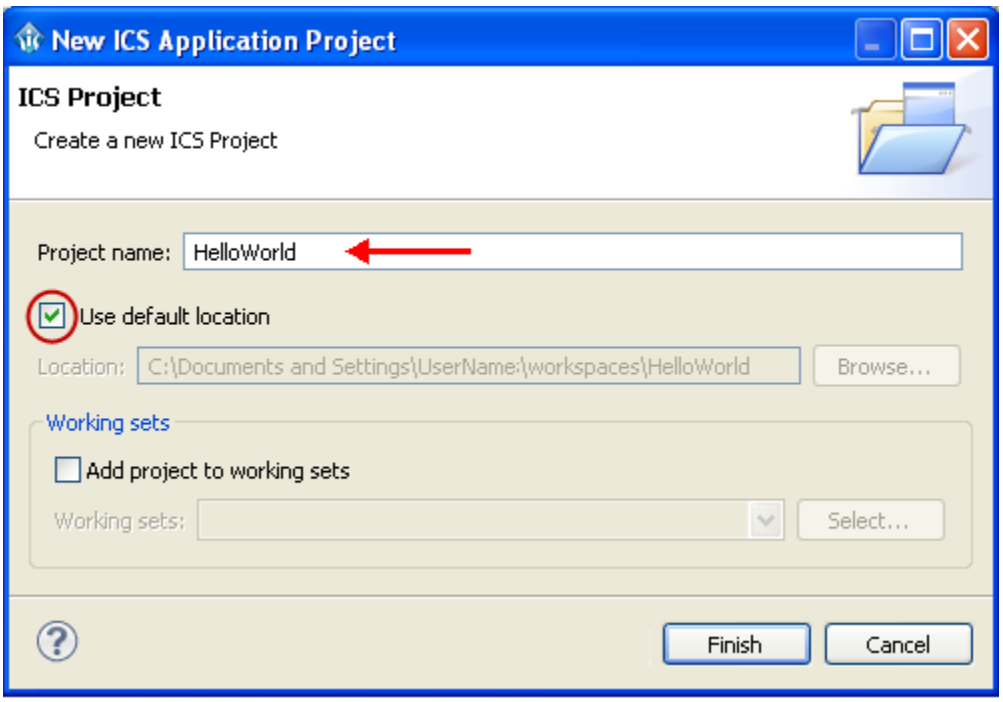


Figure 1.2 - New Project Wizard

If HelloWorld is your first IC application project, the window will not have a Next button, only Finish and Cancel buttons. If you had created other projects previously in your workspace, the wizard would have a Next button, which would allow you to link other projects and libraries to this new project. However, the basic HelloWorld application will not need any other projects or libraries. Another recipe will cover libraries and linked projects.

6. Click **Finish**.

7. In the Package Explorer pane on the left, click the plus sign (+) by the **HelloWorld** project folder to expand the tree view (Figure 1.3).

   Tip: You can close any Eclipse views (the various panes in the Eclipse window) that you're not using, for example, you may not need the Outline view for the recipes. If any of the Eclipse views get closed or rearranged, right-click the **Interaction Composer perspective** box in the upper right corner of the view, and then select **Reset**. This resets the window arrangement in Eclipse. It won't delete or change anything you have entered.

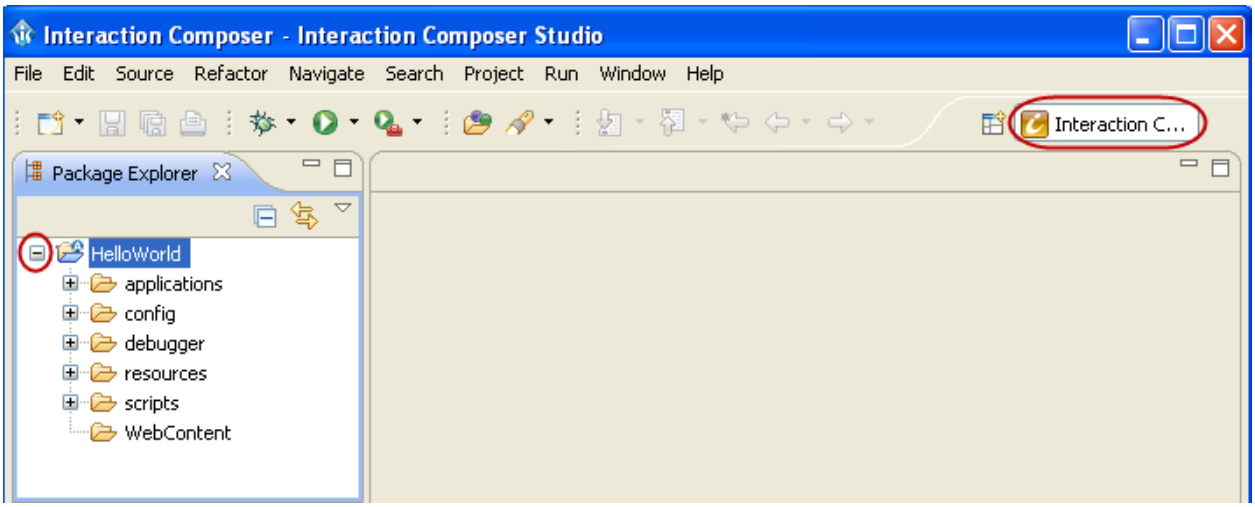


Figure 1.3 - Getting Started screen

Creating a New Voice Diagram

1. In the Package Explorer, click the plus sign (+) by the **applications** folder to expand it.
2. Right-click the **voice** folder.
3. Hover over **New**, and then select **ICS Voice Diagram**. See Figure 1.4.

   Note: Figure 1.4 has many diagram options. Several are described in other recipes.

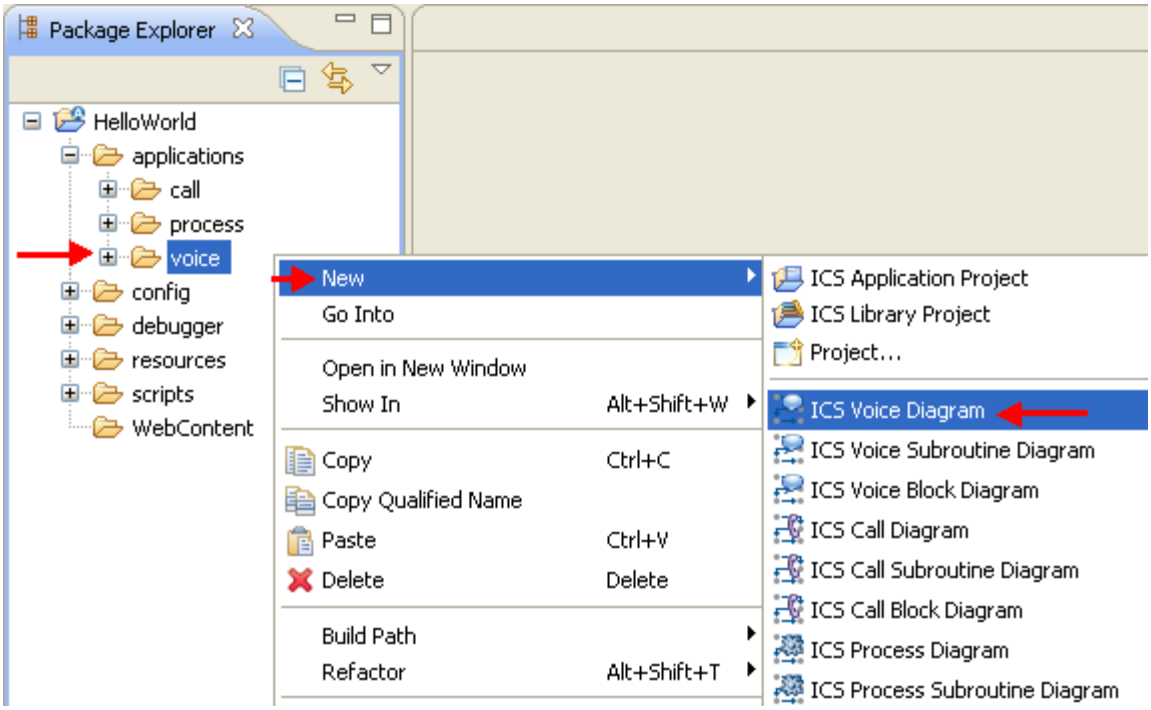


Figure 1.4 - Creating a Voice Diagram

The New ICS Voice Diagram window opens (see Figure 1.5).

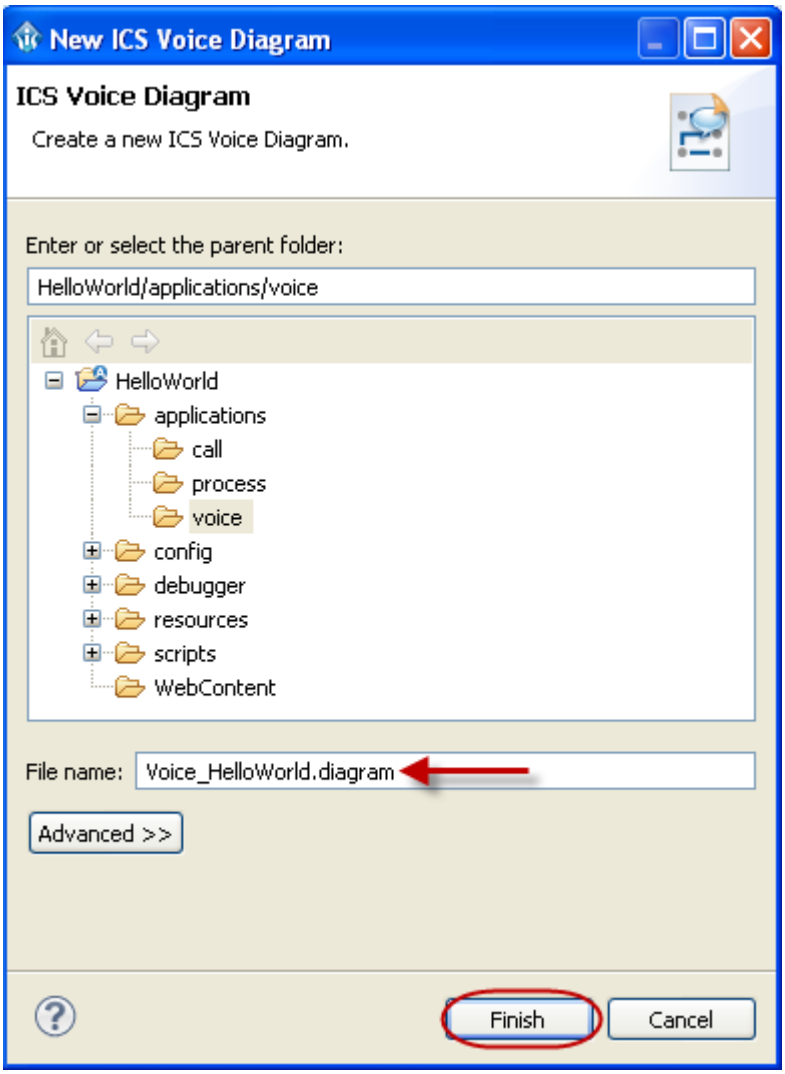


Figure 1.5 - Creating a Voice Diagram file

Note: The tree view highlights the appropriate default folder location. You can only add a diagram to a related folder: for example, add call-related diagrams to the call folder and voice-related diagrams to the voice folder. If you select another folder type, a validation error displays in the header area of the creation wizard window.

Note: In the File name field, the system creates a unique default name and adds a **.diagram** extension. You can rename the file, but you must keep the .diagram extension.

4. For this sample project, you could accept the default filename. However, you will create several Voice diagrams, so type **Voice_HelloWorld.diagram** to distinguish them.
5. Click **Finish**, and a Voice diagram flowchart opens in the Canvas view (Figure1.6).

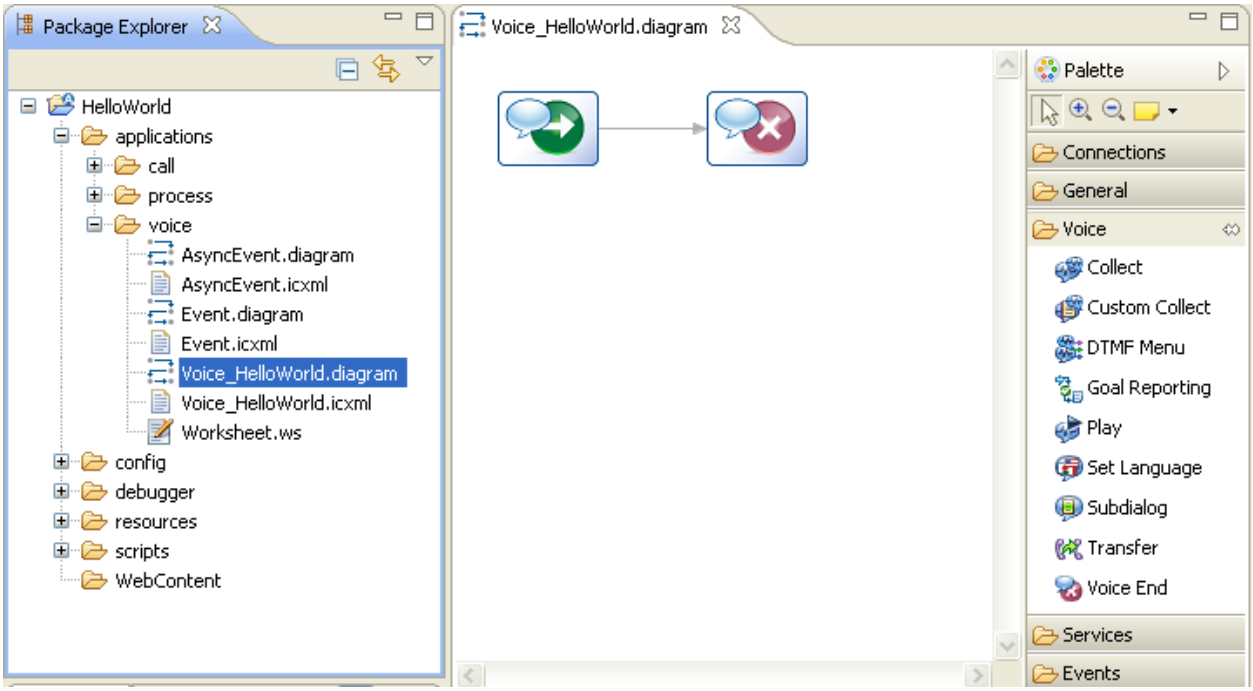


Figure 1.6 - First Voice Diagram

The Voice diagram has two icons:

- Green— represents the beginning state of the voice diagram
- Red— represents the end state of the voice diagram

To the far right is the Palette view with various drawers (Figure 1.6). Use these components to build your application flow. Since Voice diagrams deal only with the interactive voice dialog, you will use many components in the Palette's Voice drawer.

For now, you can close the other drawers in the Palette:

- Click the folder of a drawer to close it.
- Click again to open a drawer.

Note: Later, you will use other components to define functions in Call or Process diagrams, for example:

- Answering a call
- Launching a voice dialog
- Stepping through a database procedure

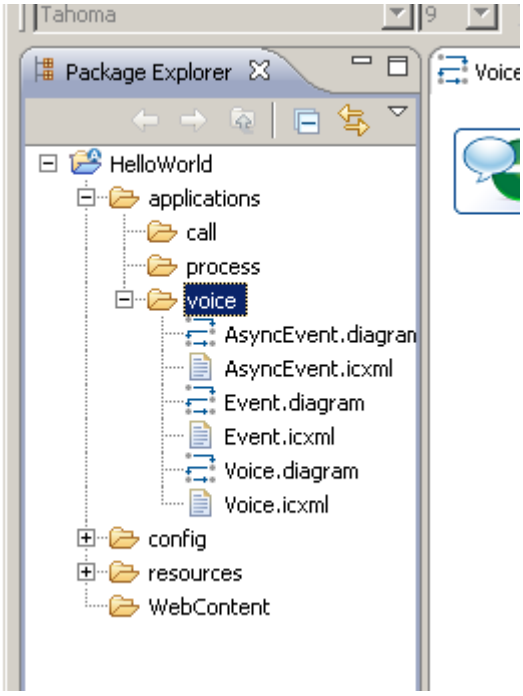


Figure 1.7 - Voice Diagram files

In the Package Explorer, expand the **voice** folder to see the files that were created when you created the Voice diagram (Figure 1.7). For information on the other files, see the *Interaction Composer User's Guide*.

Adding a Play Component to a Diagram

To add a "Hello World" prompt:

1. In the Voice drawer of the Palette, left-click the **Play** icon.
2. Release your finger and move the mouse pointer to the Voice diagram. The mouse pointer changes to a box/plus pointer, which indicates that the pointer is holding a component.
3. Move the box/plus pointer beneath the Voice End component on the diagram, left-click again and the icon is added to the Canvas at that location. (See Figure 1.8).

   Note: When you add the Play icon onto the workspace, a text box is automatically displayed near the icon.

4. In the text box, type **HelloWorld** as a label for this Play icon. (No spaces in the label.)

   Note: On the diagram, be sure to label components wherever possible since labeling helps developers understand what each component does. If you click off of a component without labeling it, the label box disappears. If you want to label a component later, select that component and begin typing. The label box will appear as you type. To edit an existing component label, select the component and press the **F2** key. This will put the selected component's label field into edit mode.

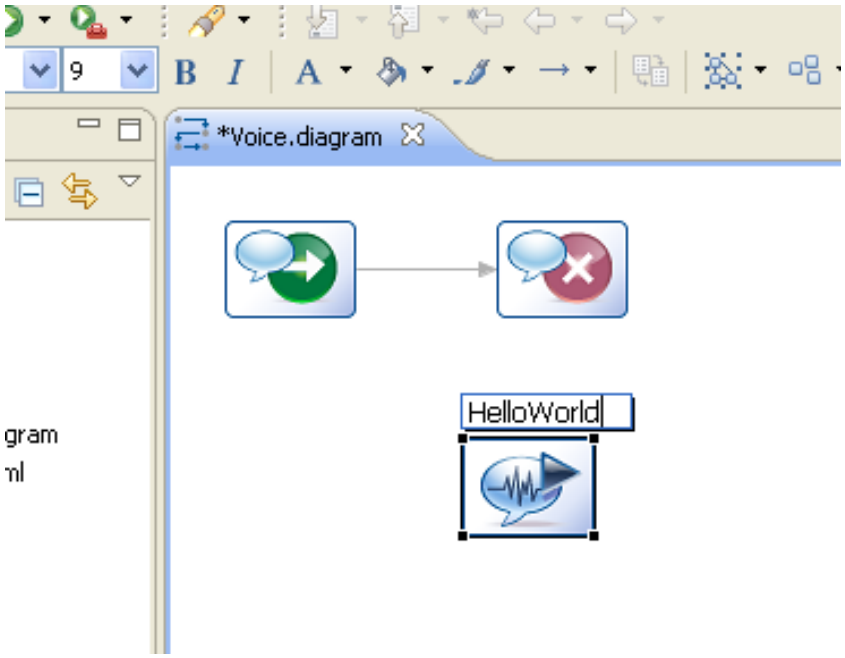


Figure 1.8 - Adding an icon

In this diagram, the first icon is Voice Begin. By default, it has a flow connection to the Voice End icon.

To reroute the dialog flow from the Voice Begin to the Play icon:

1. Click the connection pipe between the Voice Begin and End icons. Handles appear on both ends of the connection pipe.
2. Hover over the handle on the terminating end of the pipe (Voice End icon) to get cross-hairs, then click and drag the pipe to the Play icon. The pipe should now stay connected to the Play icon.

   Tip: When connecting actions, start at the originating icon and then make the connection to the terminating icon.

   This connection arrow communicates to the system that after the dialog begins, it should go to the Play icon and play the prompt.
3. Hover the mouse over the HelloWorld Play icon, and a new pipe appears.

   Tip: Do not select the icon, just hover. Hover on the side that's closest to where you want the pipe.
4. Hover over the terminating handle of that pipe, and drag the handle to the Voice End icon (red & white X with a dialog balloon). See Figure 1.9 for the completed diagram.

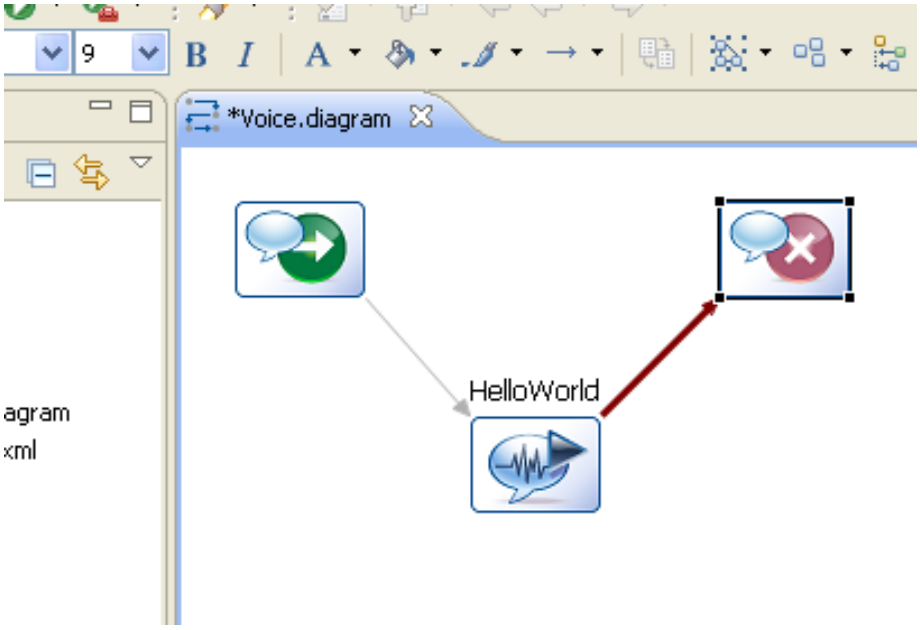


Figure 1.9 - Icons all Connected

Note: A text box appears next to the pipe after it is connected, allowing you to label that particular leg of the dialog flow. This text box only appears with newly created pipes, so dragging an existing pipe to a new connection won't cause it to appear.

For this sample project, you do not need to label the connection. Just click on the white space outside the text box.

To label an existing pipe that doesn't have a label:

a. Select the connection by clicking on it, and start typing. A label will appear.

b. Continue typing.

To edit an existing connection label:

a. Select the connection, and press **F2**.

b. Start typing to delete the old label, or click inside the blue highlighted box to insert a change.

Automatically Arranging the Diagram

When adding icons, the alignment may not be exact. To automatically align or rearrange the icons:

1. On the toolbar, click the **Arrange All** button (Figure 1.10).

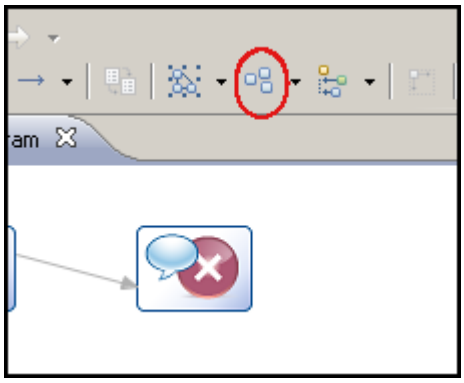


Figure 1.10 - Arrange Button

The icons align. (See Figure 1.11.)

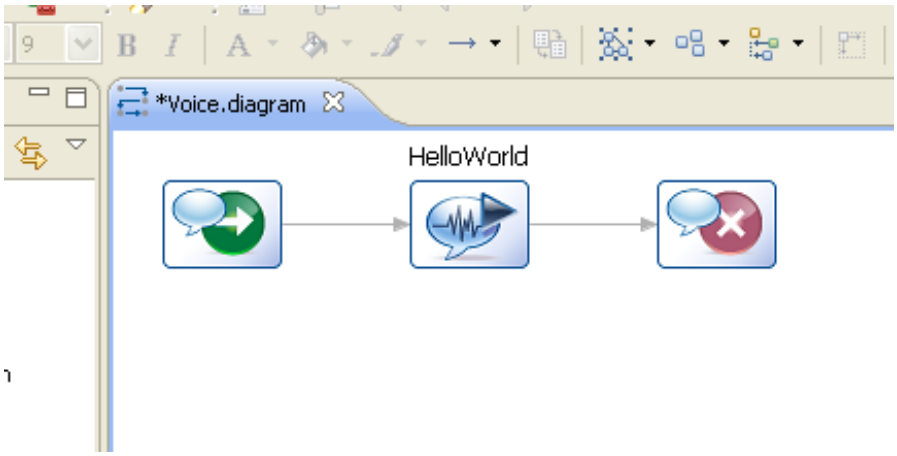


Figure 1.11 - Auto Arrangement

Prompt Types

Finally, create a prompt on the Play icon.

1. Left-click the **Play** icon and its Properties view appears below the diagram window (see Figure 1.12).

   Tip: You may need to raise the view divider above the Properties view

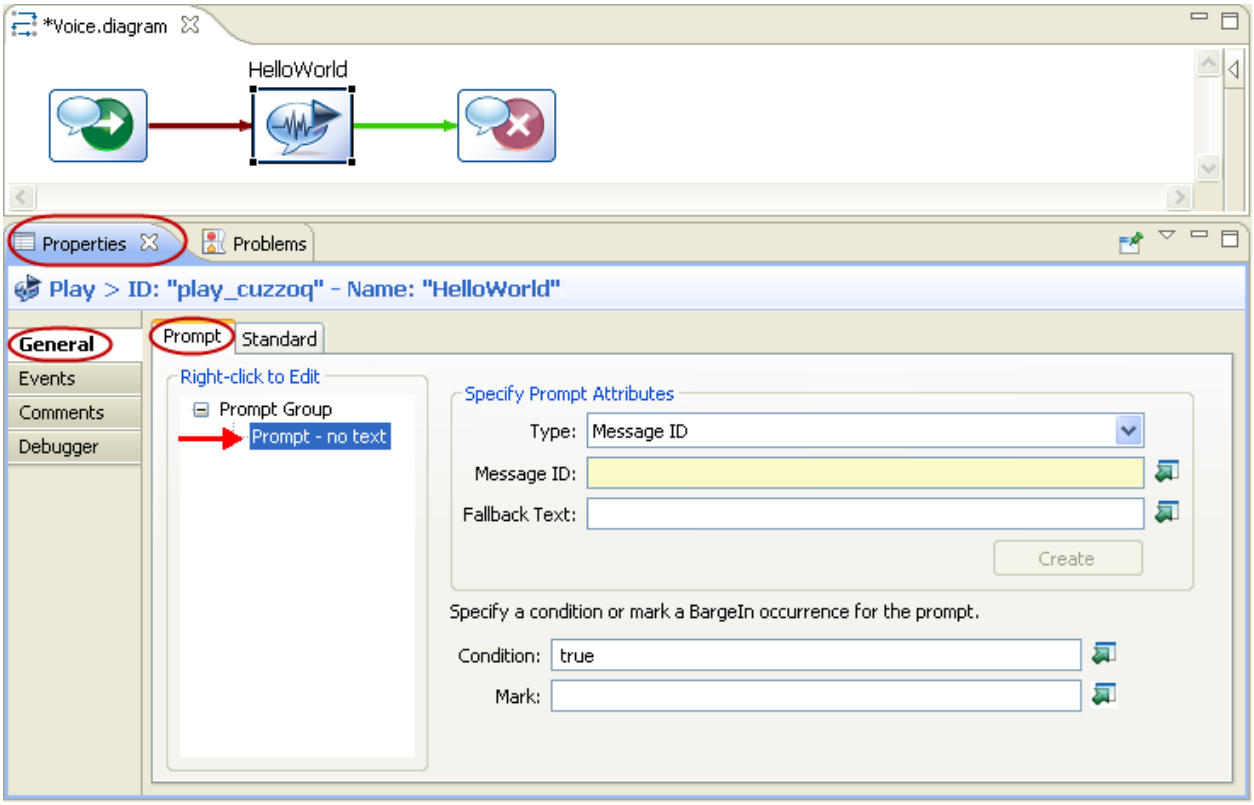


Figure 1.12 - Defining the Prompt

2. In the Properties view, be sure the **General** option is selected on the left.
3. Be sure the **Prompt** tab is selected.
4. Under PromptGroup, select **Prompt – no text**. On the right, the Specify Prompt Attributes area displays.
5. In the Type field, select **TTS**.

   Typically, prerecorded audio is better quality at runtime than text-to-speech. However, for this sample application project, you won't be recording any prompts. Therefore, specify all audio as TTS. It will be rendered by the TTS engine at runtime.

   In an actual project, TTS is useful for playing text that cannot be defined during development, such as:

   - emails
   - web news articles

6. When you select TTS, the field beneath it is labeled **Expression**. (See Figure 1.13.)
7. In the Expression field, use single quotes and type the text of the prompt phrase, **'Hello World.'** This phrase will be rendered at runtime by the TTS engine.

   Tip: To speed up the development, you can copy the above text, Hello World, and paste it into the field. However, you must manually enter the single quotes. IC gives an error when you paste from Word.

   Note: Spaces are acceptable in phrases.

   Note: The TTS text can also be a variable or expression that resolves to some text. In this case the fallback text will be used if the variable or expression fails to resolve to something that the TTS engine can resolve.

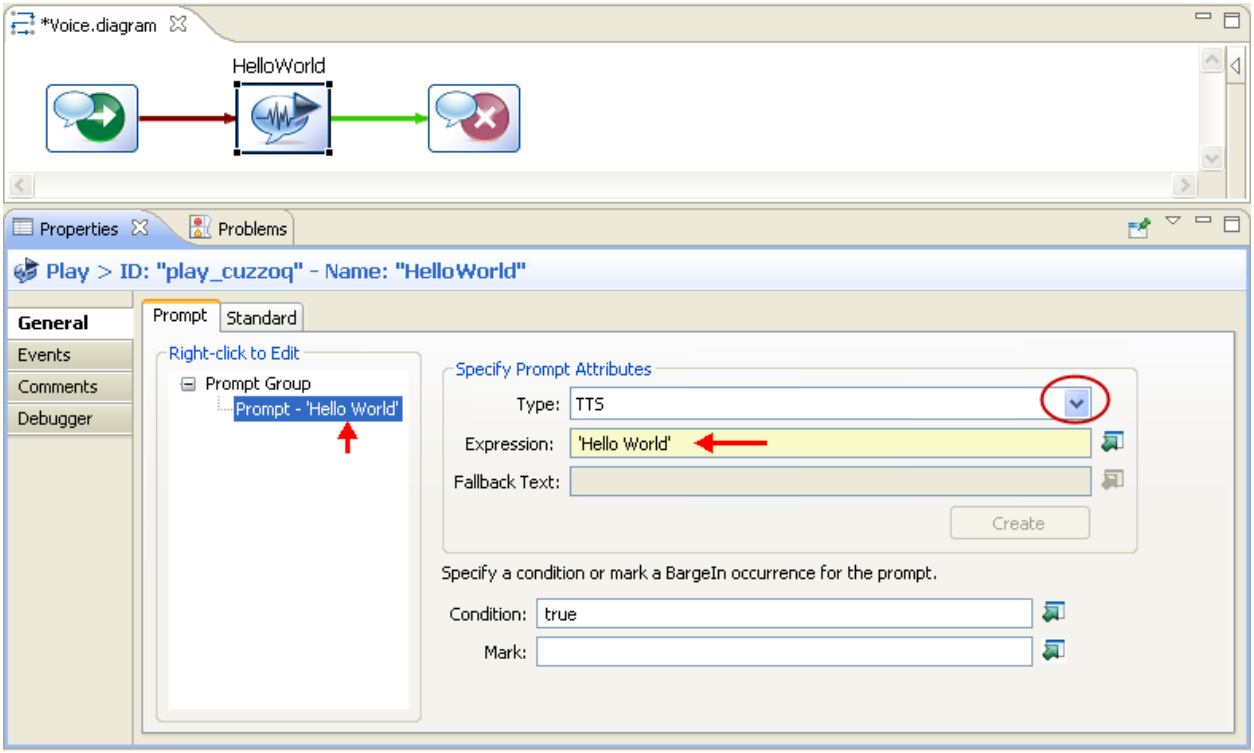


Figure 1.13 - Final Voice Diagram with Prompt Properties

8. From the IC menu bar, select **File > Save**. 'Hello World' displays under the Prompt Group.

   This saves your flowchart and all of the associated attributes, grammars, prompts, other files, and components and puts them in the appropriate folders under the HelloWorld project folder.

   Note: Click in the Voice diagram view outside the Play icon, and the Properties view will return to the overall Properties view.

   If a red X appears on a component after the Save operation, an error exists with the configuration of that component. Review the steps you made when you configured that component.

Add another Play component. Don't connect it to the other components because you'll be deleting it. This will just demonstrate how you would complete the Type field for audio messages.

For a description of all other options in the Type field, see the *Interaction Composer User's Guide*.

For recorded audio and Message ID Type:

1. Add a Play component and select **Prompt – no text**.
2. In the Type field, accept the default **Message ID**.

   Message ID indicates to the system that a prerecorded audio file exists. This is the most common option for commercial or enterprise applications because audio prompts that are recorded by a professional voice talent are more natural-sounding and more easily understood than synthesized text-to-speech (TTS) playbacks. At some point, a voice talent will need to record all the audio files in a recording studio.
3. The field beneath is labeled **Message ID**. Enter a unique value in this field to represent this audio file. This value can be a number or a .wav filename. For example, enter **1** or **hello**.

   If you have existing Message IDs and Fallback text in the Messages.msg file, you can type the Message ID in the field and the Fallback Text is automatically populated. Optionally, you can select the green launch arrow for the Message Selector dialog box and select from existing IDs.
4. In the left pane, click **Prompt – no text**. The Message ID field becomes red italic font. On the lower right, the Create button is activated.
5. At the center right, click **Create**. This logs the message ID and fallback text into the system so it can be tracked to avoid ID duplication.
6. In the Fallback Text field, use single quotes and enter the text of the message. If the audio file can't be found at runtime, the TTS engine renders the prompt as audio based on this fallback text.
7. Delete the extra Play component. You won't need it for this recipe.

To edit or delete messages that you have created:

1. In the Package Explorer, go to **<ProjectName> > resources > audio**.
2. Double-click **Messages.msg**.
3. To edit a Message ID or fallback text, click in a field.
4. To delete, select a row and click the **Delete Row** button (red X).
5. Save the project and close the **Messages.msg** tab above the Canvas view.

   Note: Optionally, you can begin by defining your messages and fallback text and then select them on a Play, DTMF Menu, or Collect component.

   Tip: When the application flowchart is completed, the IC tool can generate a list of:

   - Message IDs that you defined for each prompt
   - Fallback text that you specified for that Message ID

   You can give this list to a voice talent as an application script. The voice talent can use it to record the audio prompts in a recording studio. For more information, see the *Interaction Composer User's Guide*.

For recorded audio and URL Type:

1. Set the Type field to **URL**.

   With this option, the field beneath it is labeled **Expression**.
2. In the Expression field, specify the URL that points to where a prerecorded audio file resides.

Creating a Call Control Diagram

An IC application also needs a higher level of control that associates the voice diagram with a telephone call or other launch process. For the sample HelloWorld application, this higher level is the call control level.

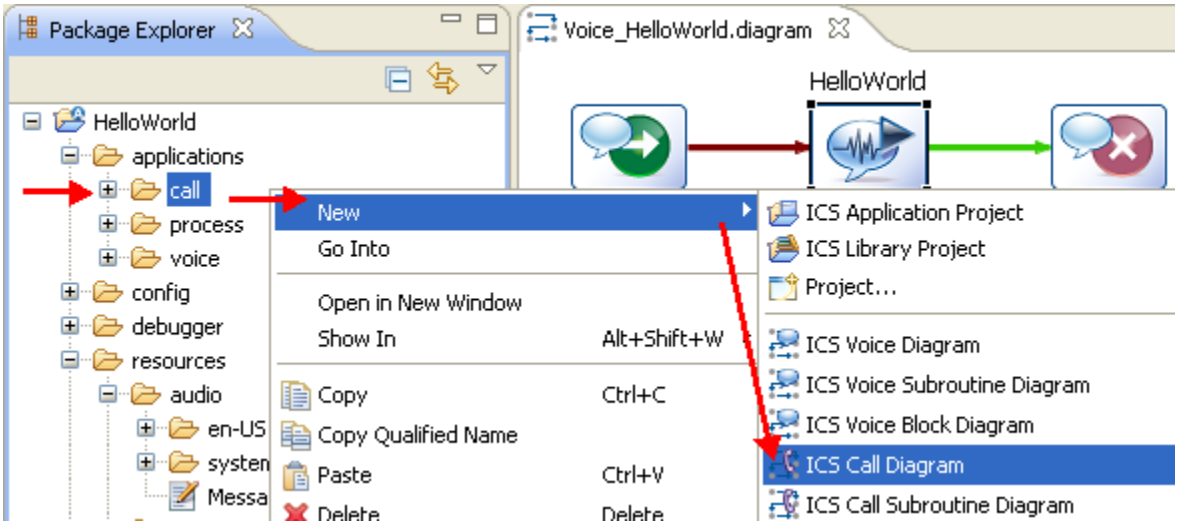


Figure 1.14 - Creating a Call Diagram

The call control diagram configures the system to wait for an incoming call. When the call arrives, the system answers the phone and starts running the voice dialog. If you want, the call control can run several voice diagrams in sequence. When the last voice dialog is finished, the system ends the call and hangs up.

1. In the Package Explorer, expand the **applications** folder.
2. Right-click the **call** folder.
3. From the list, select **New > ICS Call Diagram**. See Figure 1.14.

   The New ICS Call Diagram wizard opens (see Figure 1.15). By default, the call diagram folder is highlighted.

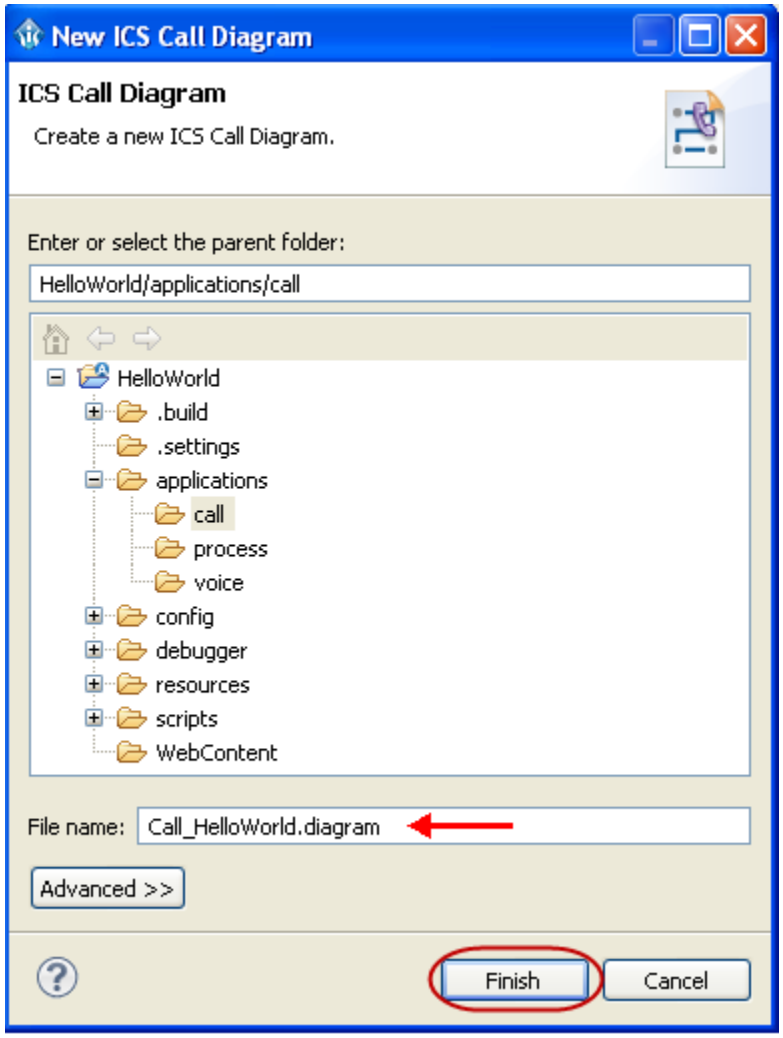


Figure 1.15 - Creating a Call Diagram

4. In the File name field, you could accept the unique default name, **Call.diagram**. However, for this sample project, type **Call_HelloWorld.diagram**.
5. Click **Finish**.
6. In the Package Explorer, expand the **call** folder. The Call Diagram wizard has created the files that define a basic default call diagram.

Linking the Voice Diagram to the Call Diagram

In the Canvas view, note that the call diagram contains four component icons representing the four basic steps of a call – call arrival, answer, voice dialog /return, and a call end. Typically, these default steps are all that most call applications require.

In the Package Explorer, notice that a red X appears by the Call_HelloWorld.diagram file, as well as the call, applications, and HelloWorld folders. This indicates that you must enter a required parameter or attribute before the application will be complete.

In the Canvas view, note that a red X also appears on the Dialog icon, which is the third icon from the left. That indicates where you must add the required parameter.

To clear up the error condition in the call control diagram, you must configure the Dialog component to indicate which voice diagram to execute after answering the incoming call. For the other components, leave the default configurations.

1. Left-click the **Dialog** component. The Dialog Properties view displays. See Figure 1.16.

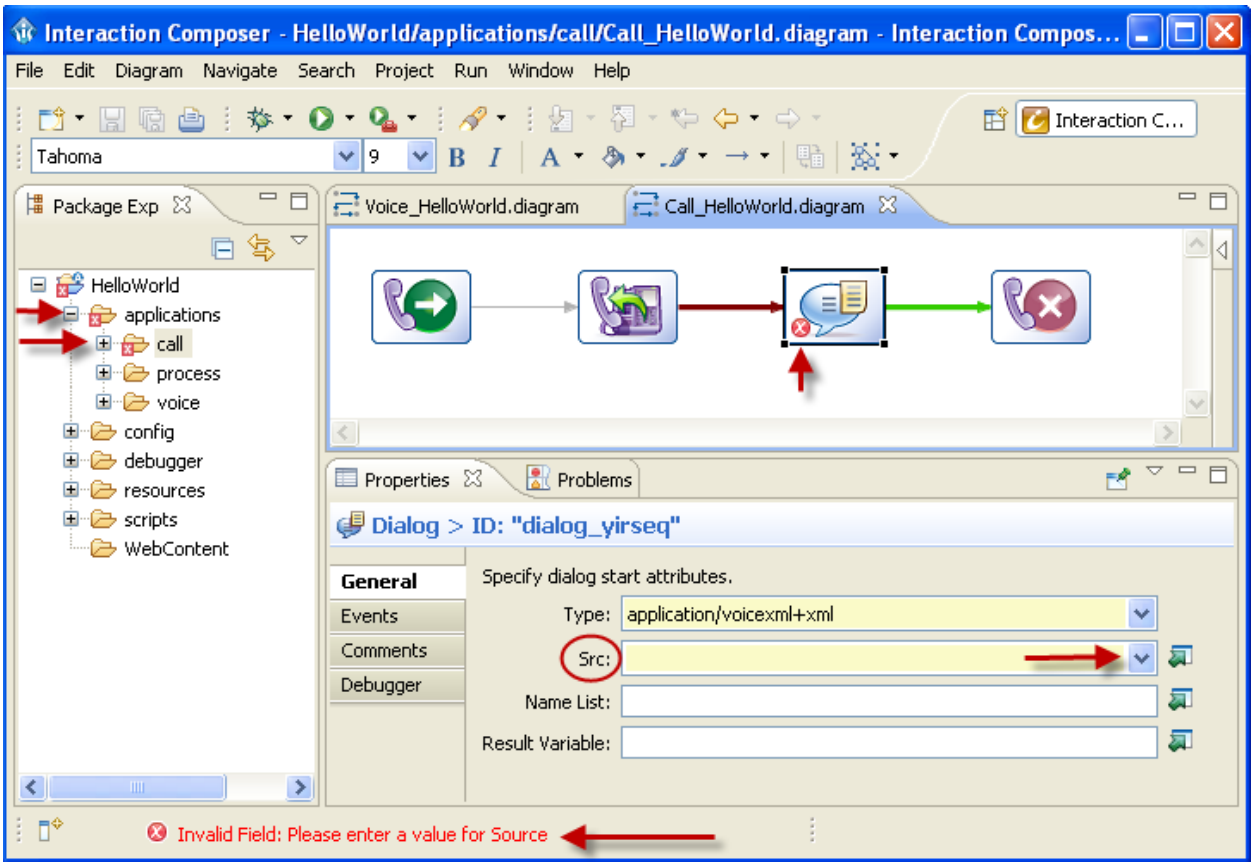


Figure 1.16 - Call Control Diagram

2. In the Properties view, be sure the General option is selected.
3. In the Type field, accept the default.
4. In the Src field (for Source), click the down-arrow.

   A list of available voice diagrams is displayed. For this sample project, you have created the Voice_HelloWorld project, so that diagram name should be displayed in the list.
5. Select that **Voice_HelloWorld.diagram** option.
6. From the menu bar, select **File > Save**. The error condition (red X) should clear.

When the voice dialog completes, control will return to the call diagram voice icon and then exit to end the call.

Note: The Canvas view now has two tabs above its window: one for the Voice diagram and one for the Call diagram. You can view either of them by clicking on the tabs. If the **Getting Started** tab is still there, you can close it.

Packaging your Application

Now, you can bundle the application with the voice diagram and call control diagram into a .WAR file and deploy it. WAR stands for Web Application aRchive, and this WAR file contains all the components that are

needed to run your voice application on an application server. When you are ready to test your application, deploy this WAR file to an application server.

To package all your application files:

1. In the Package Explorer, right-click the **HelloWorld** project folder.
2. In the middle of the list, select **Package ICS Application** (see Figure 1.17).

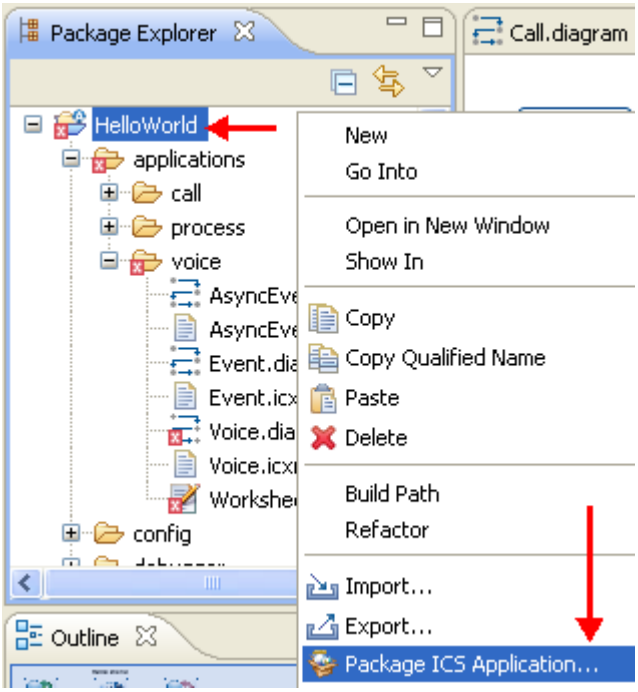


Figure 1.17 - Packaging Your Application

The Package ICS Application wizard opens.

3. In the Project field, accept the default, HelloWorld, and click Finish. In the Package Explorer, a new Deployment Artifacts folder is created. IC puts all your diagrams, projects, and WARs into the Workspace folder (HelloWorld) that you specified when you first opened IC.
4. Expand the Deployment Artifacts folder. It contains the HelloWorld.war file.

Recipe 2: DTMF (Touch-Tone) Menus

Approximate time: 35 minutes, depending on prior experience.

What you will learn with this recipe:

- Using the DTMF Menu Component
- Defining DTMF Menu Options
- Understanding the Default Path
- Handling Simple Errors
- Leading and Trailing Prompts
- How Events Work

This recipe explains how to build touchtone menus with branching, using a sample news-weather-sports application project. When a caller calls into this DTMF automated system, the system prompts the caller with options for news, weather, or sports. To simplify the coding, this recipe uses a touchtone keypad for the menu selections instead of spoken commands: the caller can press 1 for News, 2 for Weather, and 3 for Sports. After the caller selects an option, the system will state which service was selected, thank the caller, and exit.

Note: Later recipes explain how to use WSDLs to access web services for news, weather, and sports. The WSDLs extract the appropriate text from a website, which the text-to-speech engine then plays to the caller.

Creating the Project

To keep your workspace simple, collapse the HelloWorld project folders in the Package Explorer and close any diagram tabs above the Canvas.

1. Click **File > New > ICS Application Project**. The New ICS Application Project wizard opens (see Figure 2.1).

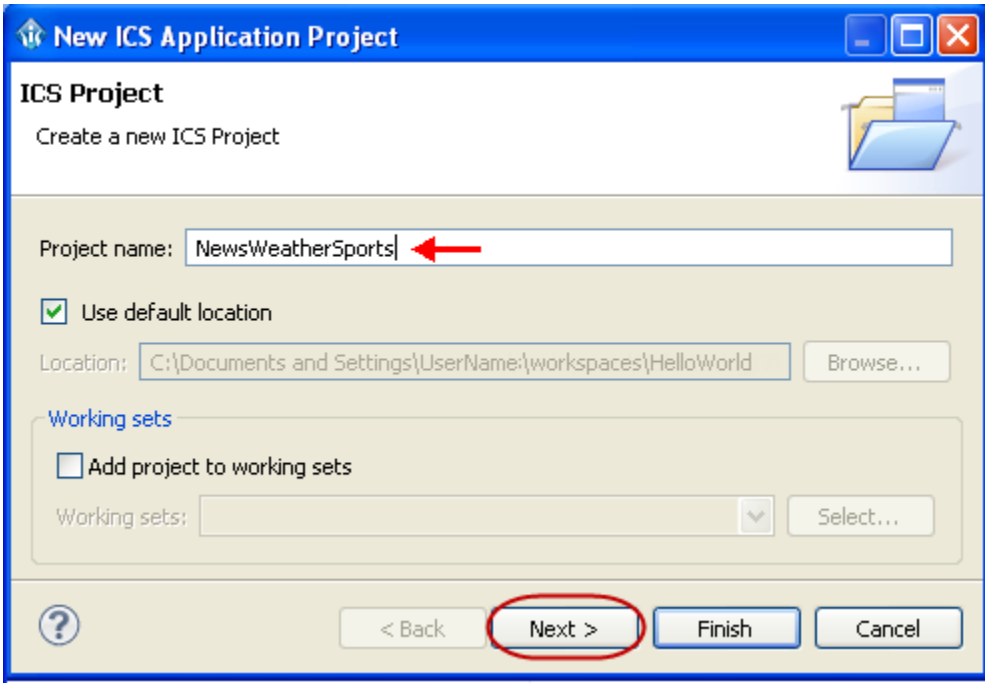


Figure 2.1- Create a new ICS Project - News Weather Sports

2. In the Project name field, type your project name, for example, **NewsWeatherSports**.

Remember that there can't be any spaces in the project name, so News Weather Sports with a space between the words is not allowed.

3. Be sure that the **Use default location** checkbox is checked. This adds your project to your default workspace folder along with the HelloWorld application project.

   Note: A Next button displays on the New ICS Application Project wizard that wasn't there when you created the "HelloWorld" application. When you create a library project, its name will be available in the next window in case you want to link the application project to a library project. However, the **NewsWeatherSports** project doesn't need any external routines. Click **Next** to see this window; then, click **Back** to return to the main wizard screen. Another recipe will explain how to create a library project and link to it.

4. Click **Finish**. The NewsWeatherSports project is added to the Package Explorer (see Figure 2.2).

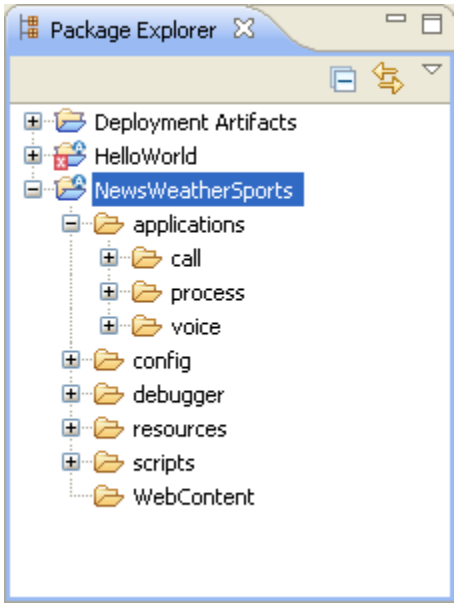


Figure 2.2 - Starting to build the News Weather Sports application

5. In the Package Explorer, expand **NewsWeatherSports** and then expand the **applications** folder
6. Right-click the **voice** subfolder.
7. Hover over **New** and then click **ICS Voice Diagram** in the list. (See Recipe 1, Figure 1.4, except that the additional NewsWeatherSports folder will now be in the Package Explorer.)

Building the Voice Diagram

When you click **New > ICS Voice Diagram**, the New ICS Voice Diagram wizard opens.

1. You could accept the default Voice name, but to help distinguish diagrams, type **Voice_NWS.diagram**.
2. Click **Finish**. The **Voice_NWS.diagram** flowchart opens in the Canvas (see Figure 2.3).
3. On the right, in the Palette, expand the **Voice** drawer.
4. Select the **DTMF Menu** component, and add it to the workspace.

   Tip: Left-click the component, drag the mouse to the appropriate location on the Canvas, and left-click again to add the component to the diagram window.
5. When you add the DTMF Menu component, a label text box displays. Label the DTMF Menu component with a descriptive name such as **NewsWeatherSportsMenu**.
6. From the Voice drawer, add a **Play** icon to the right of the DTMF Menu component and label it **News**.
7. Add another **Play** icon and label it **Weather**. Add a third **Play** icon for **Sports** (see Figure 2.3).

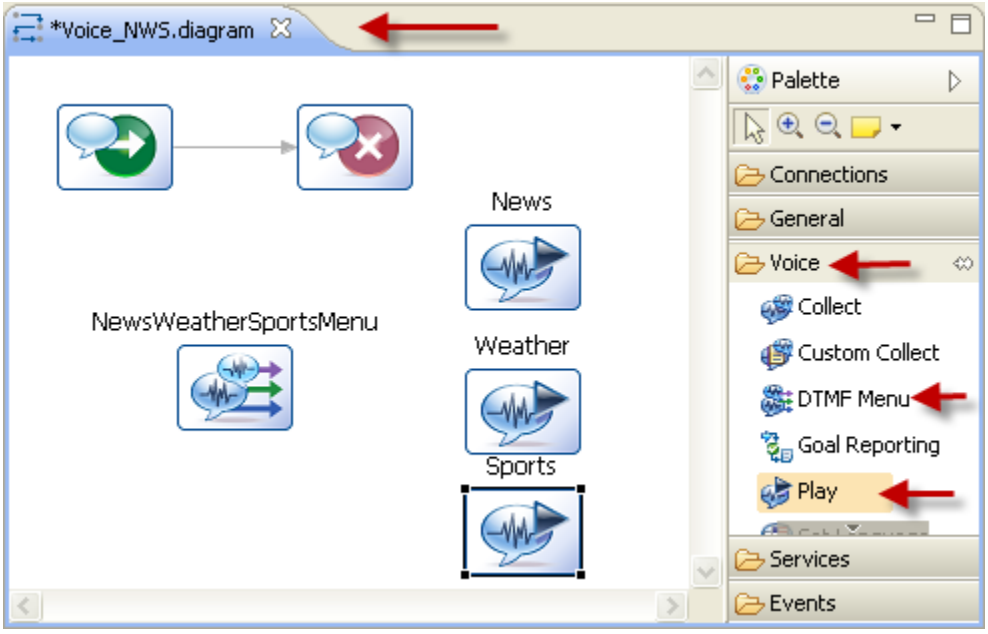


Figure 2.3 - News Weather Sports initial layout

Before you can connect the components, you must define the DTMF Menu component's menu selections and properties.

Defining the DTMF Menu prompts

To define the DTMF Menu component options:

1. Design the prompts that the system will play to the caller. This sample project has:
    - Greeting: 'Welcome to the News, Weather, and Sports access line.'
    - Menu options: 'Press one for the latest news, press two for the current weather, and press three for sports.'
    - Closing prompt: 'Please make your selection now.'
2. Add the phrases to the appropriate components in the call flow diagram.

The DTMF Menu component has many important functions. You can add all three sample prompts to this single component. It also provides options for error handling.

By default, barge-in is activated. As soon as the caller presses a DTMF key, the system will transition to the option the caller selected without playing the remaining menu options or any trailing prompt.

Typically, the menu would have navigational selections, such as: "Press four to hear the menu again" and "Press five to exit." However, this introductory recipe will just have basic options. Even if the caller doesn't specifically end the call, the key entry timeout will eventually end it anyway.

Overview of DTMF Menu Properties

Here's an overview of the contents of the **Properties** view > **General** option:

- **Menu** tab– Defines each menu item that will be played to the caller and a group name for the items.
    - Each menu item will have an associated DTMF key
    - Each menu item will indicate whether that particular menu item is active (this value could be a variable if you are building dynamic menus).
    - In the Prompt Attribute area of the menu item, the Type field indicates whether the prompt is in a prerecorded audio file or text that will be rendered by a TTS engine. If an audio file is to be used for a prompt, you can specify the filename or message ID for that audio file. In addition, you can define fallback text for TTS rendering of each audio file if the required prerecorded message is not found at runtime.
    - In the System Message field, specify a system message to be played. For example, a system prompt of the word "Press" could be prepended to a prompt that plays "one for News."
- **Prompts** tab – Defines three additional types of prompts that can be spoken in association with the main menu selection prompts. You can use audio files or TTS rendered text.
    - **Leading Prompt** –When the menu is executed, this prompt plays **before** the system lists the menu options. For example, the system might play: "Welcome to the News, Weather, and Sports line" before it plays, "Press one for…."
    - **Trailing Prompt** – Played **after** the last menu option is played. For example: "….Press 3 for Sports." Then the trailing prompt is played: "Please make your selection now."

      Note: If the caller makes a menu selection and barge-in is activated (which it is by default), all unplayed menu options, as well as the trailing prompt, will be skipped.
    - **Event** – Allows you to specify prompts to be played if an error prompt is specified and either of the two main error events (NoInput or NoMatch) occurs.
- **Standard** tab – Provides the developer with more options for how a menu is presented, including error retry limits, barge-in status, and slot filling. These options are defaulted to the most commonly used values; for this sample project, accept these defaults.

- **Properties** tab – Allows you to set properties of the ASR engine, DTMF detector, TTS engine, or Audio playback engine used by the system. These properties include “NoInput” timeouts, recognition confidence levels, and other signal processing engine parameters. In addition, you can add and set new properties specific to a certain engine. For this sample project, accept these defaults.

Adding Pre and Post-Menu Prompts

1. In the diagram, left-click the **NewsWeatherSportsMenu** DTMF Menu component. The Properties view displays below.
2. On the left, be sure that the General option is selected.
3. Raise the divider bar between the Canvas and the Properties view.
4. In the General option, select the **Prompts** tab.
5. Select the **Leading** subtab.
6. Right-click in the edit area, and an option displays.
7. Select **Add a new leading prompt group** (see Figure 2.4).

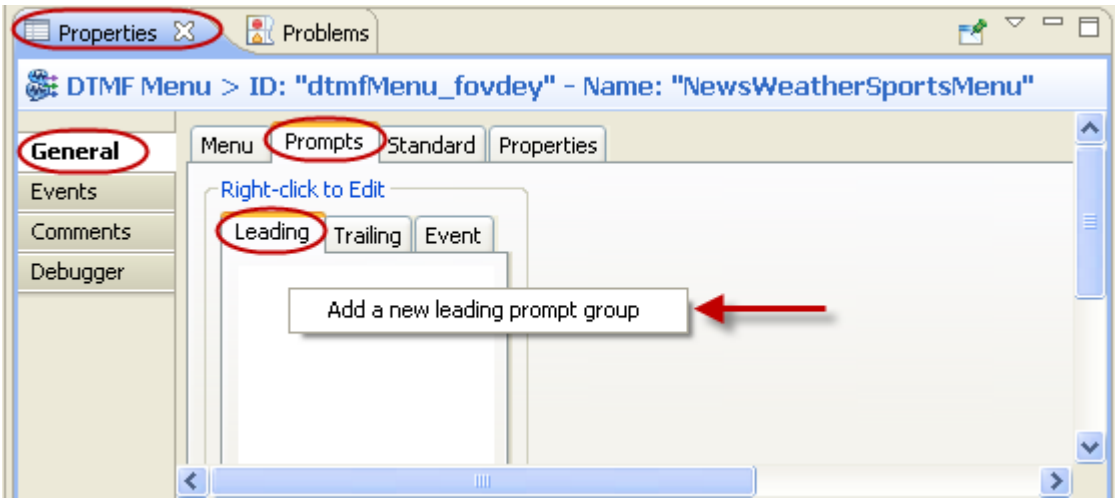


Figure 2.4 - Adding a leading prompt group

Note: The edit area adds a Prompt Group - 1 instead of just a Leading Prompt because this allows different leading prompt lists to be played, depending on some variable such as the caller’s status or capabilities.

You can add several leading prompts to the prompt, and all the leading prompts in the list will be played before starting the menu selection list.

8. In the Leading area, select **Prompt – no text**.
9. In the Type field on the right, select **TTS** (see Figure 2.5).

   Note: Typically, speech is preferred. However, to save time on this sample project, TTS allows you to deploy and test the application project without having to record the prompts. The prototype deployment is much easier.

   For information on all the DTMF Menu component properties, see the *Interaction Composer User’s Guide*.

10. In the Expression field, use single quotes and type the leading prompt text **‘Welcome to the News, Weather and Sports access line.’** (See Figure 2.5).

Tip: You can copy the above text and paste it into IC, but manually type the single quotes as the curly font causes an error in IC.

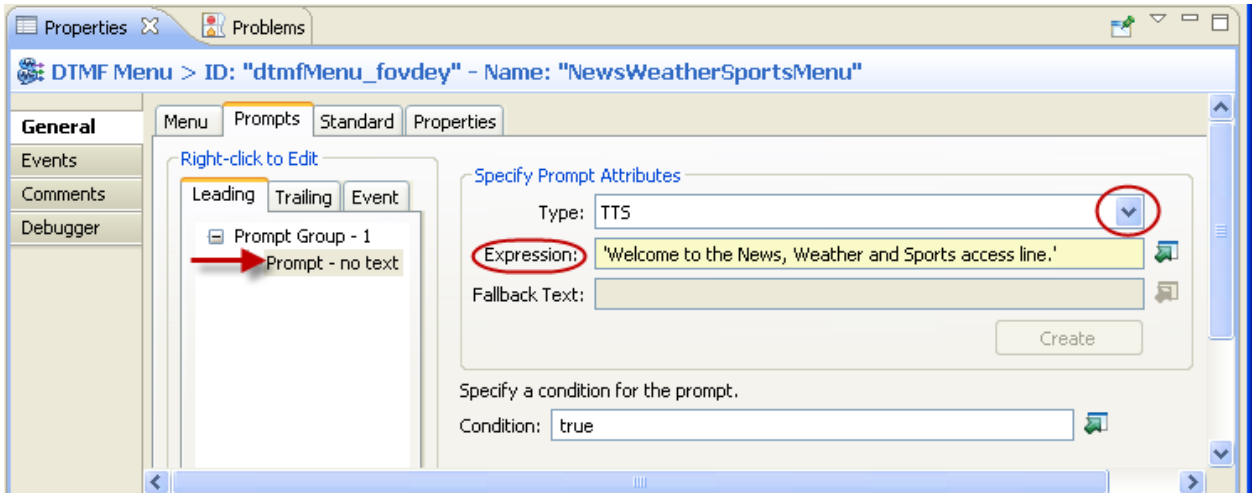


Figure 2.5 - Leading Prompt Properties

Note: When you click **Prompt – no text** or save the project, the Expression text displays in the Leading area.

Note: The Fallback Text field is not activated for TTS. It is only used if the prompt is an audio file, and the system can't locate the audio file. Then the system will use the fallback text to render TTS.

Note: The Condition attribute field allows the developer to select which set of leading prompts will be played, when under control of an external variable.

Note: The leading prompt list is played only once. If the overall menu is reprompted because the user didn't make a selection (NoInput error) or selected an invalid entry (NoMatch error), the replay will skip the leading prompt and start with the menu selection prompts.

For a Trailing prompt:

1. In the left pane, select the **Trailing** subtab.

   Note: The trailing prompt is always played after the menu selection prompts even when there is an error reprompt.

2. In the Trailing area, right-click to get the options.
3. Select **Add a new trailing prompt**. (similar to Figure 2.4 except that you select Trailing tab).

   Note: A Prompt – no text entry is added to the Trailing edit area. The trailing prompt does not support a prompt group, so you cannot define a list of trailing prompts, just a single trailing prompt.

4. In the Trailing area, select **Prompt-no text** (see Figure 2.6).

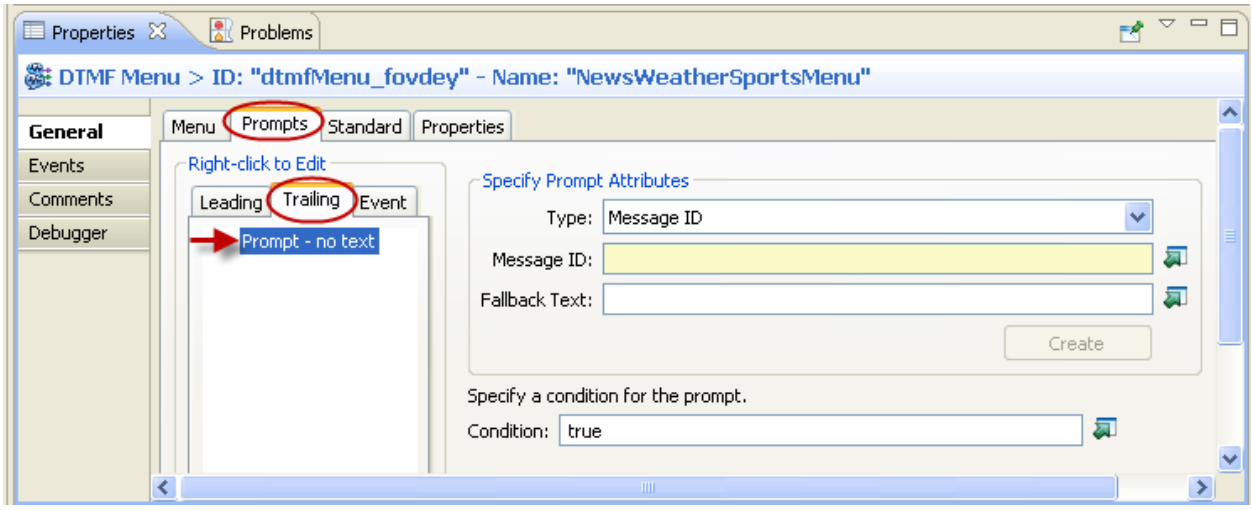


Figure 2.6 - Trailing Prompt Properties

5. In the Type field, select **TTS**.
6. In the Expression field, use single quotes and enter **'Please make your selection now.'** Figure 2.7.

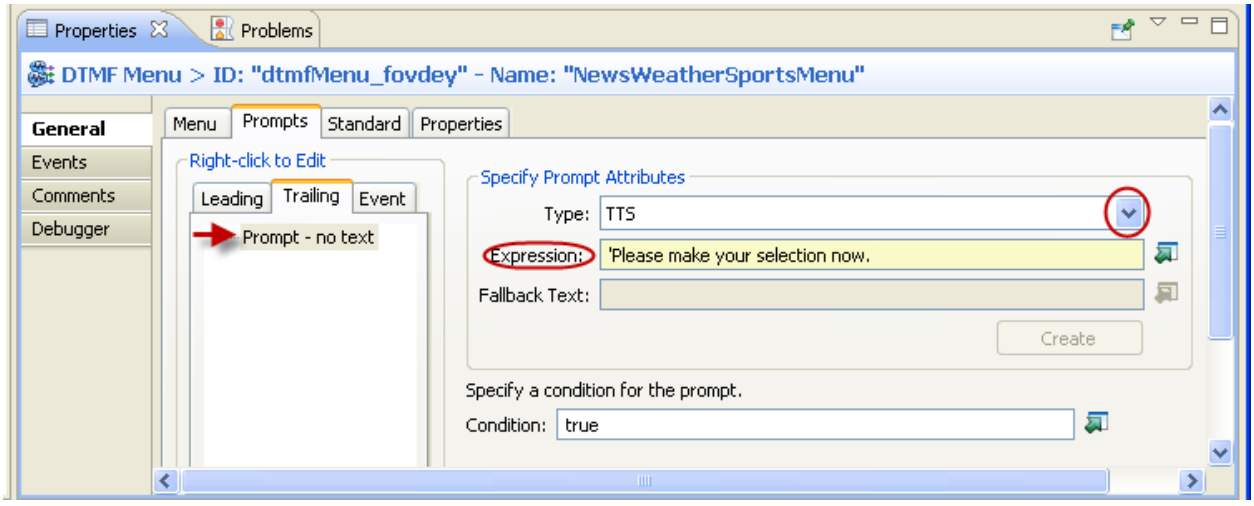


Figure 2.7 - Trailing Prompt Properties

7. Save your application project.

In the lower right, a Condition variable defined with each trailing prompt still allows external program control of which (if any) trailing prompt will be played.

This completes the design of the leading and trailing prompts.

Adding Menu Prompts

To define the three main menu prompts:

1. In the DTMF Menu's Properties view > General option, click the **Menu** subtab.
2. Be sure that **Menu Group – no name** is selected.
3. In the Menu Group Name field, type **News Weather Sports**. (You can copy and paste this name.)
4. Left-click **Menu Item – no name**.
5. In the Type field, select **TTS**.
6. In the Expression field, use single quotes and enter **'Press one for News.'**
7. In the Menu Item Name field, enter a unique name, **News**.
8. In the DTMF Key field, select **'1'** from the list. (See Figure 2.8).

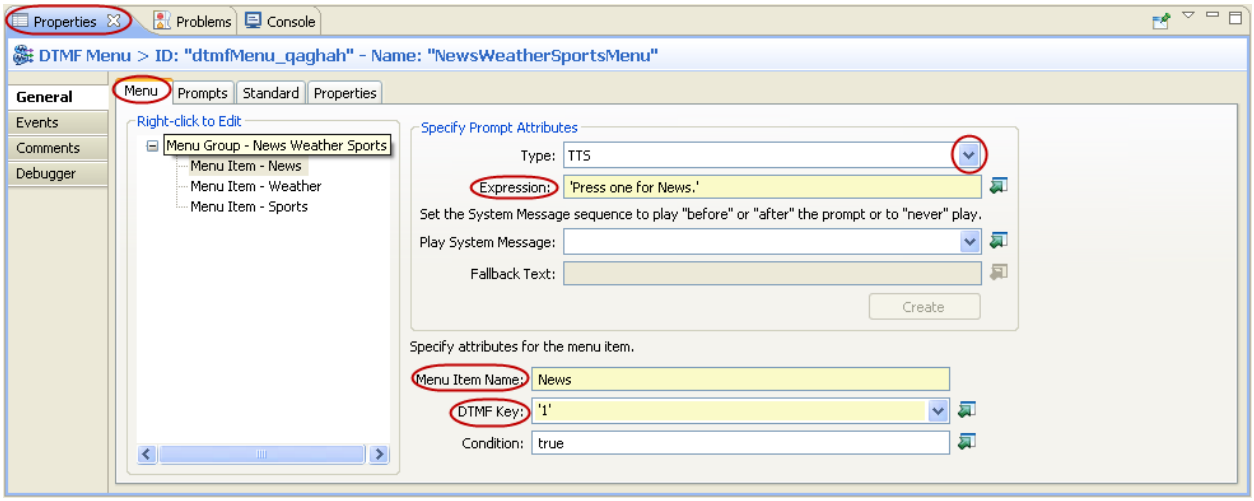


Figure 2.8 - Creating Menu selections

9. Right-click **Menu Group** and from the options, select **Add a new menu item**.
10. Left-click **Menu Item – no name** and configure the following:
    - Type—**TTS**
    - Expression field—use single quotes and enter **'Press two for Weather.'**
    - Menu Item Name— **Weather**
    - DTMF Key field—**'2'**
11. Right-click **Menu Group** and from the options, select **Add a new menu item**.
12. Left-click **Menu Item – no name** and configure the following:
    - Type—**TTS**
    - Expression field—use single quotes and enter **'Press three for Sports.'**
    - Menu Item Name— **Sports**
    - DTMF Key field—**'3'**

Connecting DTMF Menu Exit Paths

To connect the component icons to represent the dialog flow:

1. Click the connection pipe between the Voice Begin and End icons.
2. Hover over the handle on the terminating end of the pipe (Voice End icon) to get cross-hairs, then click and drag the pipe to the DTMF Menu icon.
3. Select the DTMF Menu component and then hover the cursor over it until an attachment connector appears.
4. Drag the DTMF prompt connector handle to the **News** Play component. A pop-up menu displays with two options (see Figure 2.9).

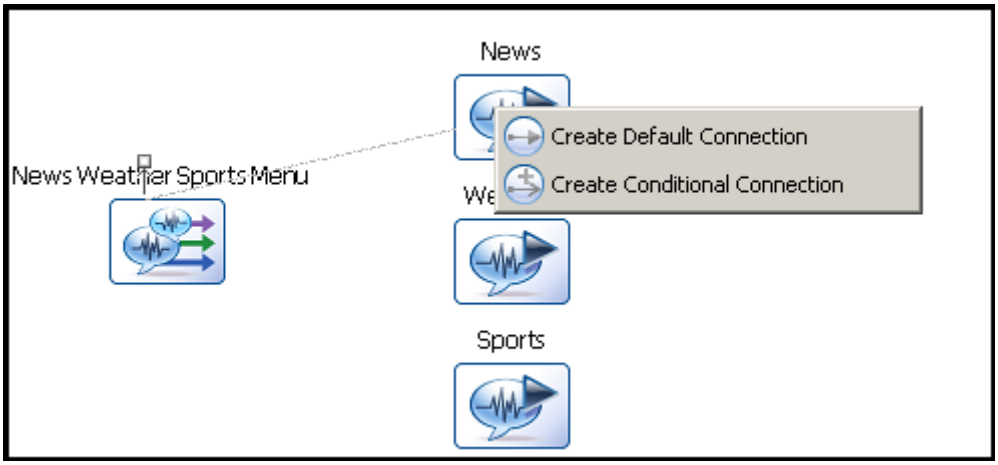


Figure 2.9 - Connecting menu selections

5. Select **Create Conditional Connection**. A second pop-up menu displays.

   Note: Typically, use the Default Connection for error exits or other conditions out of the normal dialog flow. For more information see a later recipe or the *Interaction Composer User's Guide*.

6. For the News Play component, select **News** from the pop-up menu. (If you don't see the News pop-up menu, confirm that you created all the menu prompts on the previous page.)
7. Hover the cursor over the DTMF Menu component until an attachment connector appears.
8. Drag the DTMF prompt connector handle to the **Weather** Play component.
9. Select **Create Conditional Connection**.
10. Select **Weather**.
11. Repeat steps 5 through 8 using the Sports branch.

    Note: As each branch is connected and you select one of the pop-up Menu Item names, that name is removed from future pop-up lists.

See Figure 2.10 for the current call flow diagram.

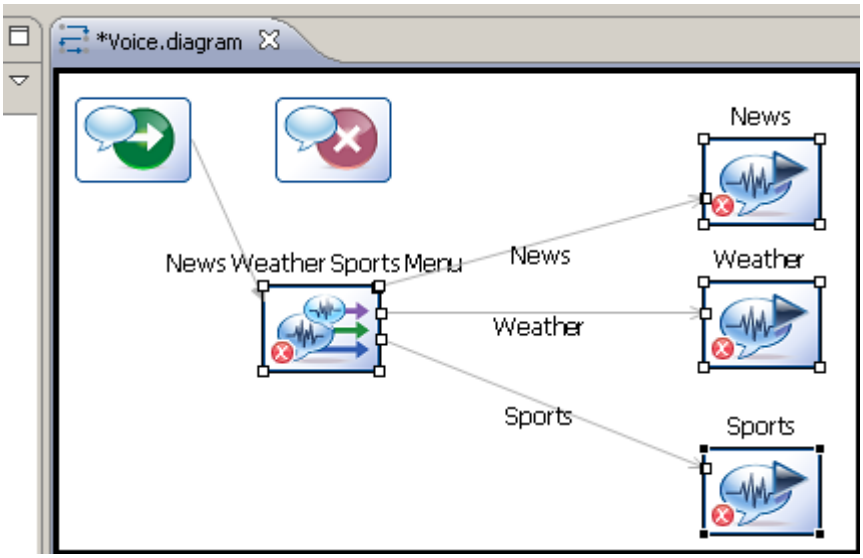


Figure 2.10 - Connecting the Play Legs

Configuring the NewsWeatherSports Play Components

To configure each Play component with a prompt message that informs the caller which option was taken in the dialog flow:

1. Select the **News** Play component > **Prompt – no text**, and set the TTS to play **'You have reached the News application.'**
2. Select the **Weather** Play component and set the TTS to play **'You have reached the Weather application.'**
3. Follow the same steps for the **Sports** Play component.
4. Add a new **Play** component to right of the three previous components, and label it **Goodbye**.
5. Configure this component to play **'Thank-you for using the News, Weather, and Sports service. Goodbye.'**

   Note: For now, the application won't access actual live news sources. That will be covered in a more advanced section of this cookbook. This call flow just informs the caller which menu leg was taken.
6. Connect each News, Weather, and Sports component to the **Goodbye** component. You don't need to label the connectors.

   No matter which service is selected, the Goodbye message will play after any selected service has completed.

Playing Goodbye

To connect the rest of the diagram:

1. Drag the red Voice End icon to the far right of the Goodbye component.
2. Connect the **Goodbye** component to the **Voice End** component. You don't need to label the connector.

   This will end the dialog after the Goodbye prompt.
3. Save the project to clear all but one of the errors. The diagram should resemble Figure 8.11.

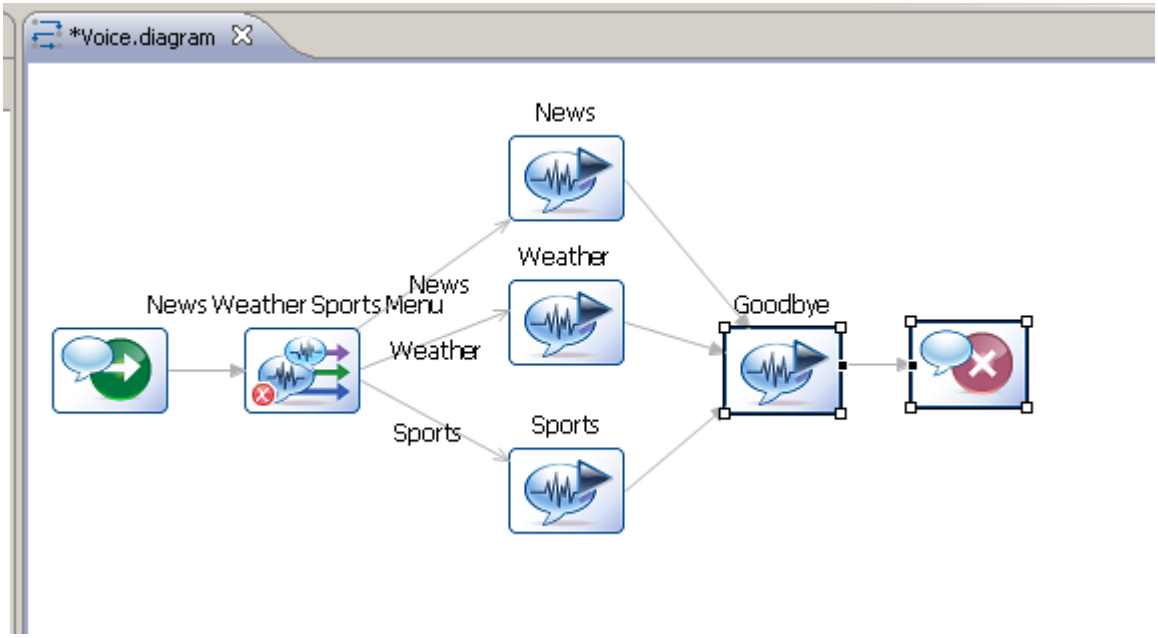


Figure 2.11 - Connecting the dialog

Diagram Errors

If you've configured all the components correctly, only one red X should remain on the diagram's DTMF Menu component.

To investigate any problems in the diagram:

1. Beneath the Canvas, select the **Problems** view (next to the Properties view).

   Note: This view lists all problems in a diagram, no matter which component is selected in the diagram.
2. Expand **Errors**. Two required features of the DTMF Menu are missing: **default** and **resultVariable**.
3. In the diagram, click in the white space or on another component.
4. In the **Problems** view, double-click one of the **Error** rows. The DTMF Menu component is selected in the diagram. To correct these two errors, see the next section.

   If another component has a red X on it, one of its required attributes may not be configured correctly.

Handling Menu Errors

To fix the **resultVariable** error:

1. Select the **DTMF Menu** icon in the diagram, and then click the **Properties** view.
2. Select the **Standard** tab.
3. Enter a variable name in the Result Variable field, for example, **NWSresult**. This recipe won't use the value, but naming the variable removes an error flag from this component. Typically, this is flagged when the variable that holds the menu option that the caller pressed has not been defined.
4. For this basic recipe, accept the default settings. See Figure 2.12.
   - MaxErrors—When set to **1**, and 1 input error occurs, the call flow exits the DTMF Menu component and takes the **Default** path towards the Goodbye component. (See steps below.) No error prompts or reprompts will be played. This isn't the best dialog design, but, for now, it keeps things simple. A later recipe will handle more user-friendly error recovery schemes.

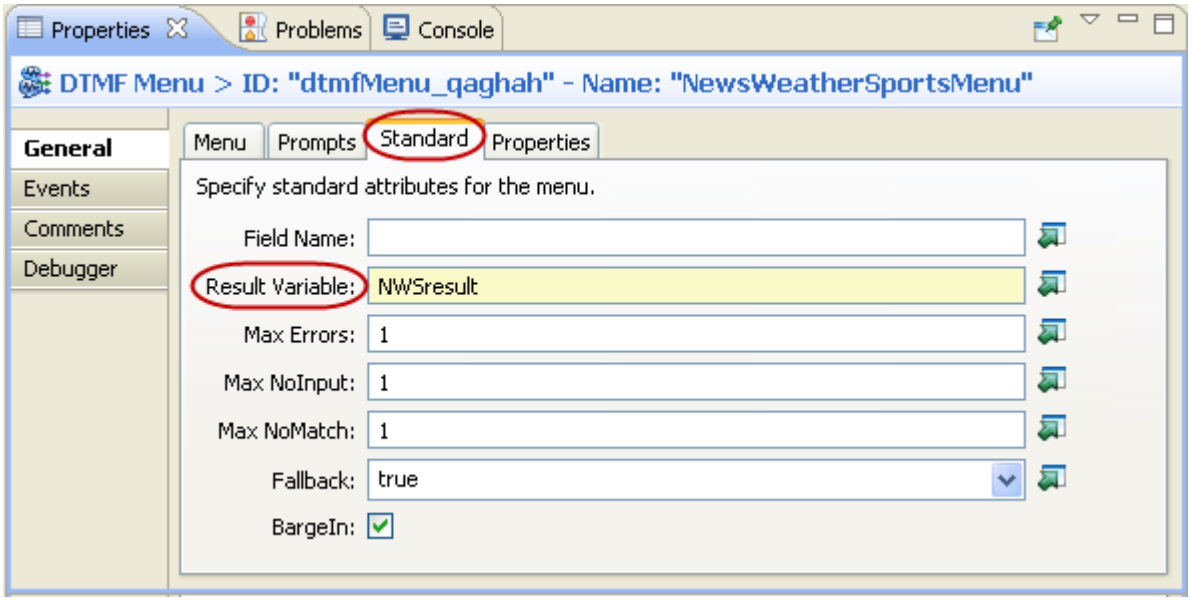


Figure 2.12 - DTMF Menu Advanced Properties

To fix the **'default'** error/problem:

1. Select the **DTMF Menu** component.
2. Hover the mouse over the DTMF Menu component to get another connector handle.
3. Drag that connector handle to the **Goodbye** component.
4. At the pop-up menu, select **Create Default Connection**.
5. In the connector-label text box, type the label **DefaultError**. (Play components must have a valid play expression and be connected to a valid next dialog step.)

   Sometimes, a caller has an input error (a caller presses another DTMF key that was not an option or the caller presses no DTMF key). Or another type of error occurs. For these default errors, the system plays "Goodbye" and hangs up.

   Tip: Drag the middle of the DTMF > Goodbye **connector handle** down below the call flow.

6. Save the project. All red X's should disappear and any corrected Errors are cleared from the Problems list.

Adding Call Control

You have completed the voice dialog portion of the News Weather and Sports application. To complete the project, add a Call Control diagram to the application project. See the Recipe 1's call control call flow for an example, and then complete these steps:

1. Create a Call Control diagram: **NWS call folder > New > ICS Call Diagram**.
2. In the wizard's File name, type **Call_NewsWS.diagram**.

   Note: The file name must be unique, but you could accept the default **Call.diagram** name. The Hello World project also had a **Call.diagram**. Because these are two separate projects with their own separate call folders, the **Call.diagram** name would be unique to each application project.

   However, since you have several Voice and Call diagrams in this sample project, typing a descriptive name will make it easier to differentiate the tabs.
3. Click **Finish**.
4. In the call control diagram, select the **Dialog** component (third component).
5. In the Dialog's Properties view, be sure the General option is selected.
6. In the Type field, accept the default.
7. In the Src field, select **Voice_NWS.diagram** from the list. This links the call diagram to the voice diagram.
8. Save the application project.

   All the red x's should be gone in both the call diagram and the folders.

8. When the red x's are gone, package your project as a .war file. See the steps in the Hello World recipe. However, in Package Explorer, be sure to select the **NewsWeatherSports** root folder instead of HelloWorld.
9. Export the runtime WAR file so you can test this recipe.

Recipe 3: Speech-Enabling a Touch-Tone Application

Approximate time: 25 minutes, depending on prior experience

What you will learn with this recipe:

- Building a Menu for both Speech and DTMF input
- Building a Speech Grammar
- Using the Case Component
- Copying parts of a diagram
- Logging errors

With this recipe, you will design an application project similar to the DTMF recipe except that this recipe will handle speech input from the caller to make menu selections.

New Project Setup

To create a new speech project:

1. To simplify the workspace, close any open diagrams by clicking the X on each tab above the Canvas.
2. Click **File > New > ICS Application Project**. The New ICS Application Project wizard opens
3. In the Project name field, type **SpeechNewsWeatherSports**.

   Tip: Remember, no spaces in the project name.
4. Click **Finish**. The project is added to the Package Explorer. (See Figure 3.1.)

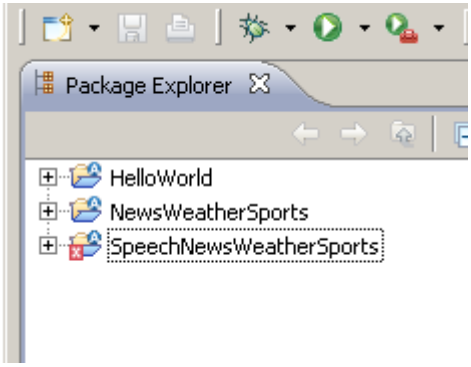


Figure 3.1 – Creating a New Project

5. Create a **Voice_Speech.diagram** for this application project. See Recipe 1's "Creating a New Voice Diagram."
6. Create a **Call_Speech.diagram**, also described in Recipe 1.
7. Point the **Dialog** component of the **Call_Speech.diagram** to the **Voice_Speech.diagram** that you just created.
8. Save the project, and the red x's should disappear.

Now, you can build a speech-enabled menu to let the user speak their selection for news, weather, or sports.

Adding components

To add components:

1. At the top of the Canvas, select the **Voice_Speech.diagram** tab.
2. From the Palette's Voice drawer, double-click a **Collect** component to add it to the call flow diagram.

   Note: The Collect component is similar to the DTMF Menu component except that the caller can speak a menu selection instead of keying it in on a touchtone phone.
3. Label the Collect component **SNWSMenu**. (This stands for SpeechNewsWeatherSports.)
4. From the Palette's General drawer, add a **Case** component.
5. Label the Case component, **SNWSSelect**.

   Note: If you forget to type a component label when you first add the component, label it later by selecting the component and pressing the **F2** key (rename).
6. Connect the **Voice Begin** component to the **Collect** component.
7. Connect the **Collect** component to the **Case** component.
8. Label this second connection **MenuResult**, as it represents the passing of the result of the menu selection to the Case component.

   Figure 3.2 illustrates the voice diagram at this point.

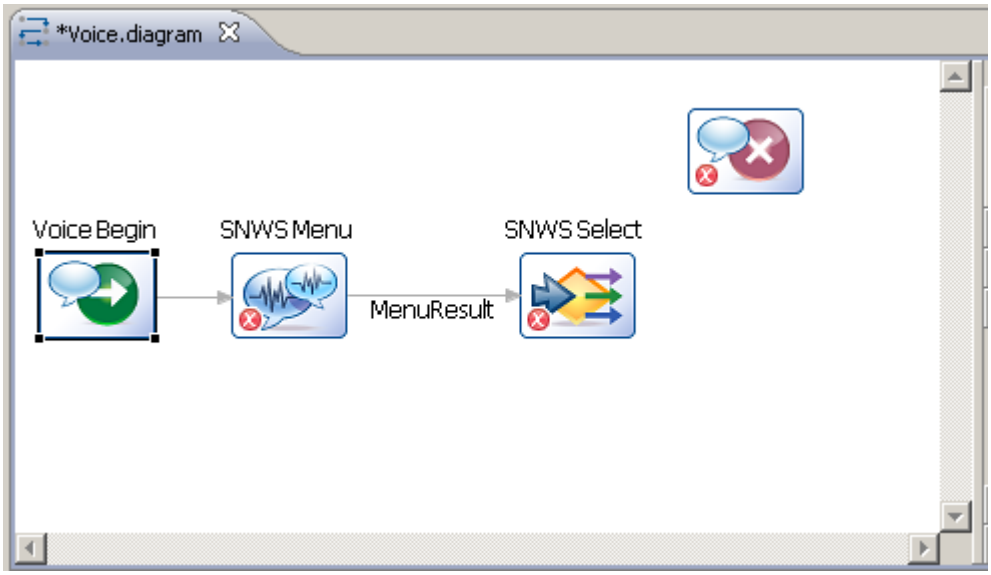


Figure 3.2 - Starting the Voice Diagram Design

This diagram represents the start of the speech-input voice script, where the system plays a menu prompt to the caller. The result of the caller's menu selection will be passed to the Case statement, which will branch execution to the appropriate task depending on the caller's selection.

Entering the Collect Component Properties

To define the properties of the Collect and Case components:

1. Click the **Collect** component to select it.
2. Beneath the diagram, in the **Properties** view > **General** option, select the **Prompt** tab.
3. Click the **Prompt – no tex**t entry.
4. In the Type field, select **TTS**.
5. In the Expression field, use single quotes and enter the prompt text: 'Would you like news, weather, or sports?'

   In a real project, you could add more Prompt Groups to handle various error conditions such as NoInput or NoMatch. However, for this sample project, leave the error handling to the default, which will exit after one error with no error prompts or menu reprompts.

Unlike the DTMF Menu component, the Collect component does not create separate output legs for each menu option. As the default, all successful menu selections as well as any error conditions exit the Collect component in the same default path after setting the result variable.

The Case component must inspect and interpret that menu result variable and branch to the appropriate task. This means, however, that there is not any separate default path for an error exit from the Collect component as you configured for the DTMF Menu component.

Error Handling in the Collect Component

At this point in the call flow, two options exist for handling errors:

- Let all errors go to the Case component along with the valid menu selections, and let the Case component determine the path. Since the result variable from the Collect component is an ECMAScript object (data structure), the case statement will have to parse the result variable and then branch to the appropriate exit.

  or

- Define a separate error exit in the Collect component by specifying Inline handling of errors in the Collect component.

  Note: IC provides additional error-handling options, such as an asynchronous error handler process. For more information, see the *Interaction Composer User's Guide*.

For this sample application project, the simplest approach is to design a separate error exit in the Collect component.

To create a separate error exit in the Collect component:

1. Select the **Collect** component.
2. In the Properties view, select the **Events** option on the left.
3. In the list of event handlers, set each event to **Inline**. (See Figure 3.3).

   Inline means that the system will handle the errors in the same diagram space as the regular voice flow, so the Collect component should provide a specific exit path for these error events.
4. Save the project.

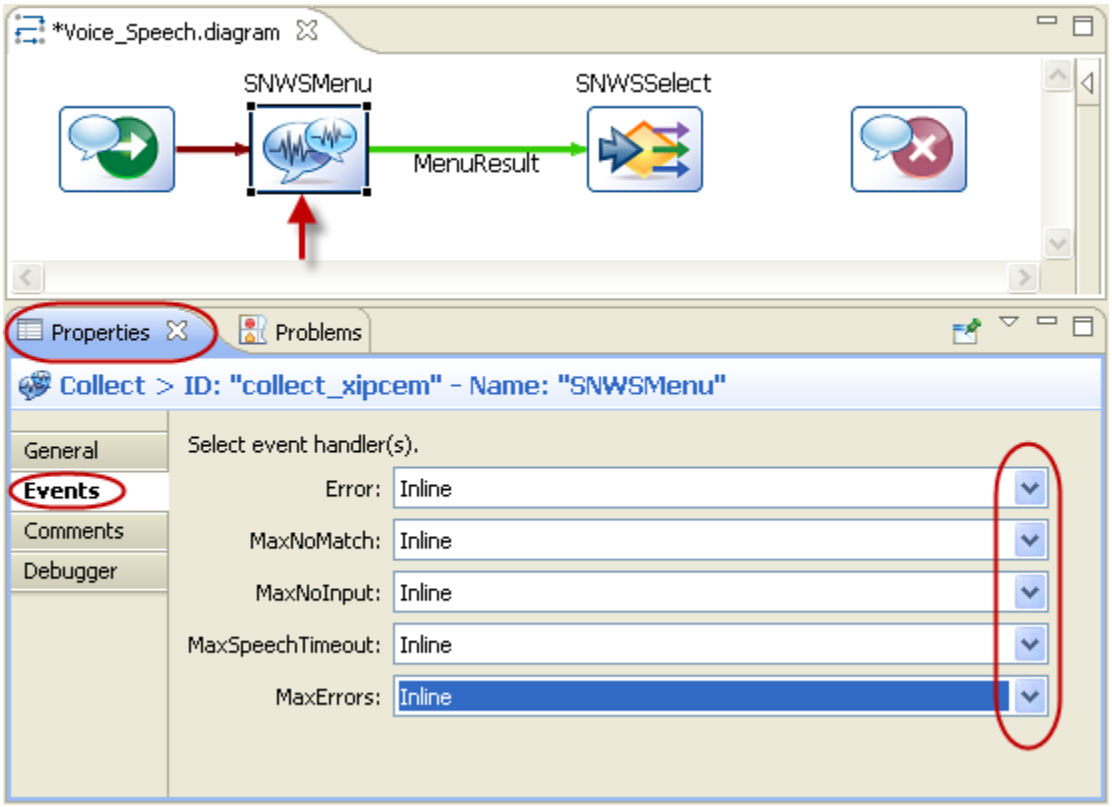


Figure 3.3 - Setting the Error Paths for the Collect Component

Now the Collect component will have two exit paths that will be made available when connecting its outputs:

- Default exit path—taken when a valid menu selection is made.
- Error path—taken when any error occurs.

Collect Component and Grammar Builder

In a speech application, callers can say anything they want in response to a menu prompt. Therefore, the voice-user-interface designer and the enterprise must design prompts that guide a caller to speak from a list of words and phrases that the speech recognition engine has been programmed to recognize for each specific prompt. This is known as directed dialog.

Typically, designers and developers try to constrain a caller's reply to a prompt by explicitly stating the available choices. For example, a prompt that plays: "Would you like your account balance or a transfer of money" produces a more predictable response from the caller than "How may I help you?"

They must also carefully design and develop grammars that can cover a range of possible caller responses to a prompt. A well-designed script generates greater speech-input accuracy.

The list of expected words and phrases is called the grammar. When callers frequently use words or phrases that aren't in this grammar, designers and developers can add them to the system.

The grammar is a text file coded in an XML format called GRXML. Grammar files (usually with a .grxml filename extension) can be stored in the following:

- Locally in the IC application or library project.
- Remotely at an alternate location on the same server or on a separate server. To access these grammars, a fully qualified HTTP-based URL that is visible to the VoiceXML Browser is required.
- In the speech engine (internal grammars).
- VoiceXML Browsers also allow grammars to be coded inline in VoiceXML.

Note: VoiceXML Browsers and speech engines that follow the W3C standards must also support a set of standard built-in grammars. IC also supports these grammars in the URL field. These built-in grammars cover both speech and DTMF input simultaneously:

- Boolean (Yes/No)
- Currency (Dollars)
- Date
- Digits
- Number
- Phone
- Time

This sample application project won't use the standard grammars. You will need to create a unique one. However, you won't need to learn the details of GRXML at this point. You can use the Interaction Composer Grammar Builder to do a preliminary design of the grammar.

This will be a very basic grammar: the single words News, Weather, or Sports, with no synonyms. Therefore, when you test the application and grammar on the phone at a later time, be sure to speak only the words and phrases that exist in this grammar that you created. A real-life grammar would include synonyms and other phrases that a caller might speak, like "I want the News" or "Give me Sports."

Note: Currently, the ICS Grammar Builder only supports single–slot grammars. You can add words, phrases, and synonyms.

After you create a grammar, you will save two files:

- Grammar Builder's representation file, with an **.ivgr** extension
- Actual GRXML grammar file, with a **.grxml** extension

To build a grammar for this application project:

1. In the Package Explorer, right-click the **SpeechNewsWeatherSports** project folder.
2. Move your cursor near the bottom of the list, and select **ICS Grammar Builder**.

   The ICS Grammar Builder is launched (see Figure 3.4).

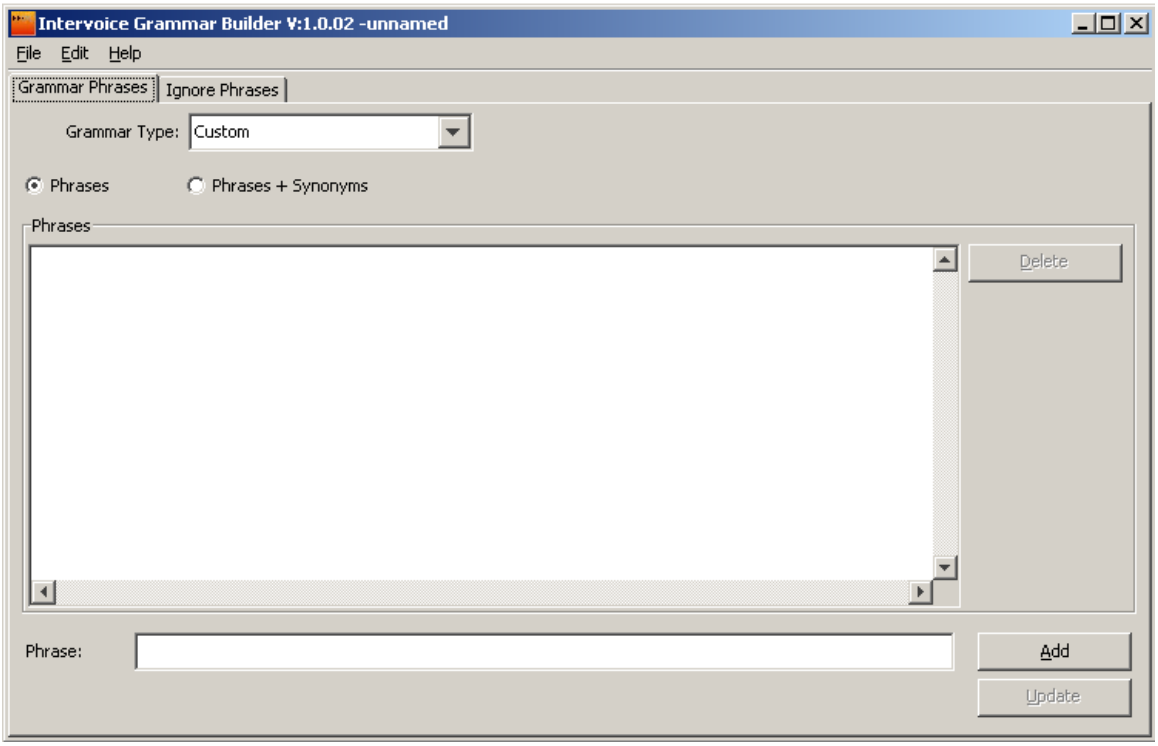


Figure 3.4 - ICS Grammar Builder

3. At the top, accept the default Grammar Type, which is **Custom**, and the **Phrases** option.
4. At the bottom, in the Phrase field, type **news**.
5. At the bottom right, click **Add** (or press **<Enter>**). The word News is entered into the grammar and into the large Phrases area in the middle.
6. Repeat steps 4 and 5 for **weather** and **sports**.

   Figure 3.5 shows the Grammar Builder with the grammar.

   Note: The Grammar Builder automatically arranges them in alphabetical order.

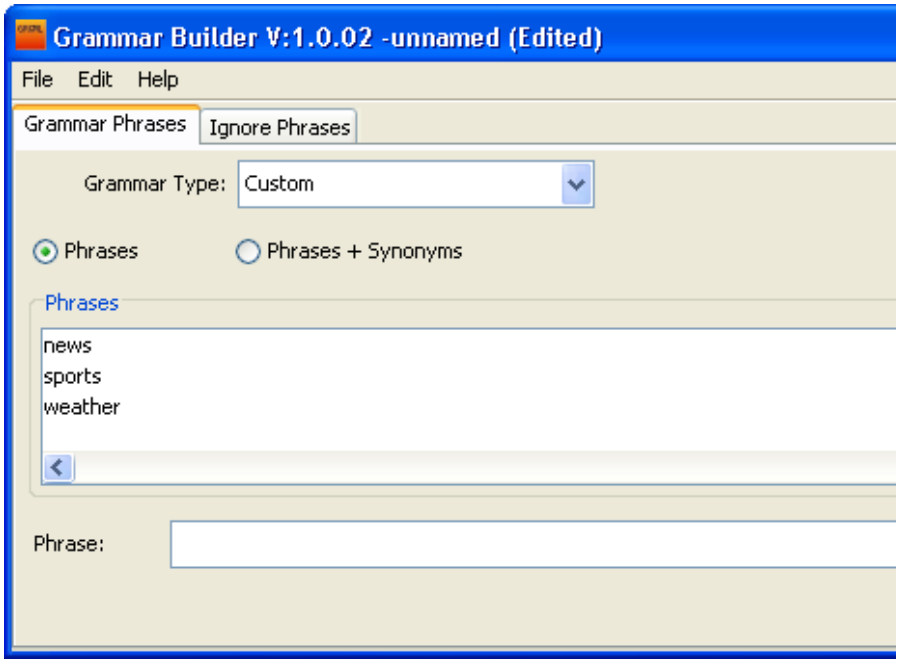


Figure 3.5 - Building a Grammar

7. In the Grammar Builder's menu bar, select **File > Save**. The Save window opens, automatically pointing to the SpeechNewsWeatherSports folder.
8. Expand the **resources > grammars > en-US** (US English) folders.
9. In the File name field at the bottom, type **snwsgrammar** for the .ivgr grammar filename.
10. On the bottom right, click **Save**.

    This will save the **.ivgr** file for that grammar. The snwsmenu.ivgr file does not appear in the IC Package Explorer. In Windows Explorer, navigate to your workspace and then to the **<projectName> resources > grammars > en-US** folder.
11. In the Grammar Builder's menu bar, select **File > Export GRXML File > OSR/Nuance 9.0**.
12. Expand the **resources > grammars > en-US** (US English) folders.
13. In the File name field at the bottom, type **snwsgrammar** for the .grxml grammar filename.
14. On the bottom right, click **Save**.
15. At a dialog box, click **OK**.

    This will save the **.grxml** file for that grammar. The snwsmenu.grxml file does not appear in the IC Package Explorer. In Windows Explorer, navigate to your workspace and then to the **<projectName> resources > grammars > en-US** folder.

    Note: To edit a previously-created grammar, launch the Grammar Builder. Then, from the menu bar, select **File > Open**. Navigate to the **resources > grammars > en-US** folder, and open the appropriate **.ivgr** file. It will open in the Grammar Builder.

Collect Component: Referencing a Grammar

You have created a custom grammar and stored it locally.

To reference it in the Collect component:

1. Close the **Grammar Builder**. (You don't need to do an additional Save.)
2. In IC, be sure the **Voice_Speech.diagram** tab is selected.
3. Select the **Collect** component.
4. In the **Properties** view > **General** option, click the **Grammar** tab.
5. In the Grammar's URL field, enter the location of the grammar. In this case, since the grammar is in the local grammar folder, you can just use single quotes and enter **'snwsgrammar.grxml'**
6. Click in the Grammar entry's Edit area or white space. The .grxml filename displays.
7. Save the project.

Collect Component: Grammar Properties

In this section, you will:

- Define the name of the variable that will hold the grammar results outcome. This will allow the Case component to determine the outcome of the menu selection. The Case component can then select the correct branch to execute.
- Define the Slot Name of the semantic slot that was filled by the caller's utterance.

  Note: Grammars can have multiple semantic slots, which allow a caller to input several pieces of information in a single utterance. For example, the utterance "I want to fly to Boston from Dallas tomorrow" has three semantic slots:

  - **from**—for example, Dallas
  - **to**—for example, Boston
  - **time**—for example, Tomorrow

  Multi-slot grammars can be useful in certain situations but they are more difficult to design, so this recipe will use a single-slot grammar.

To define the name of the variable that will carry the grammar results outcome:

1. Select the **Collect** component.
2. In the **Properties** view > **General** option, click the **Standard** tab.
3. In the Slot Name field, use single quotes and type **'slot1'**. This is the slot name automatically placed in the menu grammar by the grammar editor for a single-slot grammar.
4. In the Result Variable entry field, type the result variable, **MenuResult** (with no quotes). This defines the name of that document variable.
5. In the Input Mode field, select **'voice'** since the sample project only allows speech input and no key input. See Figure 3.6.

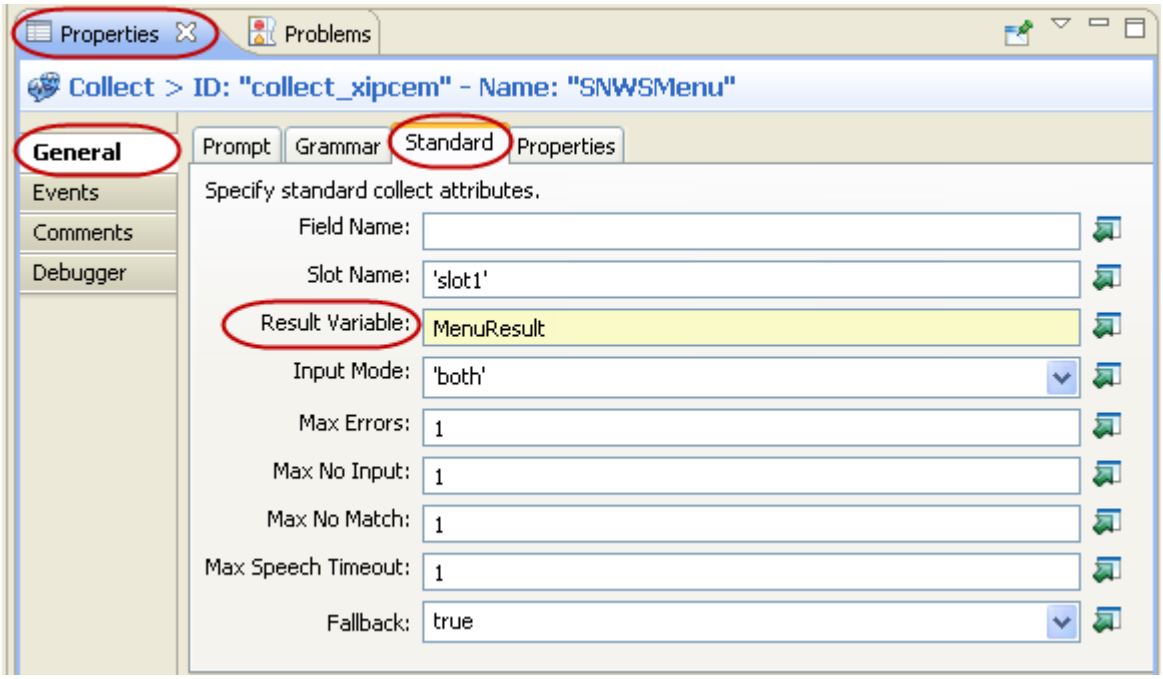


Figure 3.6 - Standard Collect Properties

Previously, when you set all the error event handlers to Inline, it created an error exit path from the Collect component. Later in this recipe, after you define some of the ending prompts, you will connect that error exit.

Configuring the Case Component

At this point, the system would collect and identify the user's selection. In this section, configure the Case component to route the selection to the appropriate Play component.

1. Select the **Case** component.
2. In the Properties view > General option > **Expression** field, enter the string **MenuResult.result** (without quotes).

   Note: This points the Case process to the specific result from the Collect component. The result variable in the Collect component was named MenuResult. MenuResult is a compound ECMAScript object (JavaScript), not a simple scalar variable. The specific place in the MenuResult object that contains the Collect result is named MenuResult.result.
3. In the Else Label field, type **Error**. This creates one exit path, labeled Error, for the Case component. If the MenuResult variable contains a value other than one of the three text strings, "News" "Weather" or "Sports", the call flow takes this exit path.
4. In the Result Equals column of the Case Values table, use single quotes and type **'news'**
5. In the Then Label column, type **News** (with no quotes).

   The Case component now has an exit path labeled "News." If the MenuResult variable matches the string "news", then the call flow takes the News exit path.
6. For the Weather and Sports exit paths:

   a. To the right of the Case Value row, click the **Append Row** button twice to add two rows.

   b. Repeat steps 4 and 5 to complete the columns with the weather and sports text (see Figure 3.7).

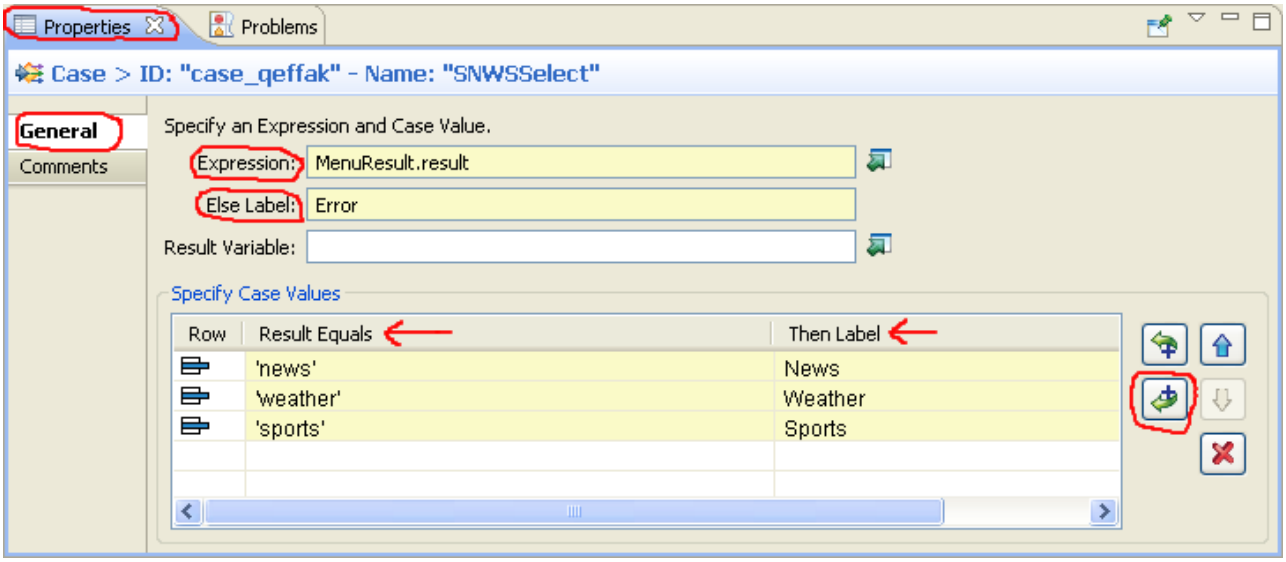


Figure 3.7 - Case component properties

.

Copying a Partial Voice Diagram

Now, you need to add:

- Three Play components to play News, Weather, and Sports.
- Goodbye Play component for the ending and error exits

Since you already designed and diagrammed that part of the call flow, you can simply copy those four Play components from the previous DTMF **NewsWeatherSports** recipe.

1. In Package Explorer, expand the **NewsWeatherSports** project. (This is the DTMF project.)
2. In **the applications > voice** folder, double-click the **Voice_NWS.diagram** file.

   Now you have two voice diagram tabs, one for the speech NWS project and one for the original DTMF NWS project.
3. Do one of these:
   - In the DTMF **Voice_NWS.diagram**, select the four Play components and their connectors.
   - Click in the white space at the edge of the four Play components and drag a select box around all four components. Be sure to leave the DTMF menu component and the Voice End component out of the selection box, as their equivalents are already in the new diagram.

   Check to make sure that the News, Weather, Sports and Goodbye Play components are highlighted. Also include the three connectors between the NWS Play components and the Goodbye component.
4. From the IC menu bar, select **Edit > Copy**.
5. Click the **Voice_Speech.diagram** tab.
6. Place the mouse pointer to the right of the Case component.
7. Right-click the mouse and select **Edit > Paste**. The four Play components, their connectors, labels, and properties are added to the Speech project.

   All the Play properties have been copied with the components.

   Figure 3.8 illustrates your current voice diagram.

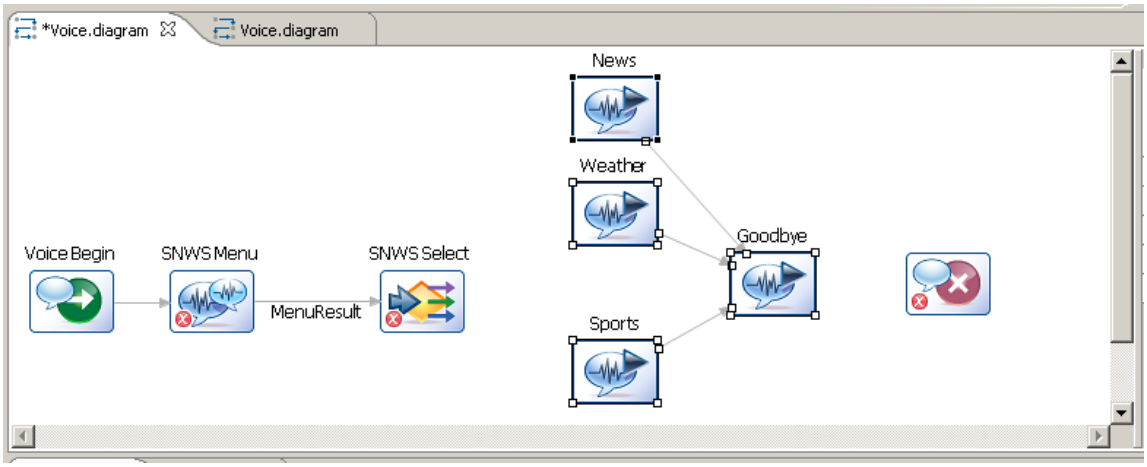


Figure 3.8 - Copying parts of a Voice diagram

Connecting the Copied Components

To connect the Play components to the rest of the diagram:

1. Select the **Case** component.
2. Hover to get a connector, then drag the connector to the **News** Play component.
3. In the connection box, click the **Create Conditional Connection** option.
4. Select the **News** option for the label.
5. Repeat steps 2 through 4 for the **Weather** and **Sports** exit paths.
6. On the Case component, hover to get a connector.
7. Drag and add the connector to the **Goodbye** component.
8. Select **Create Default Connection**. The connection is automatically labeled Error since you configured it earlier.

   The Default path is the path that any Case error (the Else clause) will take.
9. Click a white area of the Canvas.
10. Click the **Error** connector, and then drag the middle of the Error connector below the call flow to make it easier to view.
11. Save the project.

Error Handling in Collect

At this point, the easier call flow design would be to route all of the errors to the Goodbye Play component and then exit. However, the Collect component has five possible error exits: Error, NoInput, NoMatch, Speech Timeout, and MaxErrors. If the system logs a Collect error, it's helpful to know exactly which of those errors occurred. You will also need to add a Log component to track the menu errors.

To configure the system to log the menu errors:

1. In the Palette on the right, open the General drawer, click the **Log** component and release the mouse.
2. Move the cursor below the **SWNS Menu** Collect component, and then click in the Canvas.
3. Label the Log component **ErrorLog**.
4. Hover over the Collect component, connect to Log component, and select the first error condition label.
5. Repeat step 4 to connect the other four error exit paths.

6. For better readability, after you make all the connections, select a connector, hover at the middle point to get double-arrows, then drag the connectors apart and/or move the connector labels.
7. Click the **Log** component, and go to the **Properties'** view > **General** option.
8. In the Log Expression column, enter the text **MenuResult** (no quotes). This will log the contents of the MenuResult variable (ECMA object) any time a Collect error occurs.
9. Optionally, you can click the **Comments** option on the left of the Properties view, append a row, and enter a comment such as, "This is a Menu Collect Error."
10. Connect the output of the Log component to the **Goodbye** component, and label that connection **MenuError**.

    The Log component logs the error and also combines all the error exit paths into a single error path to help clean up the diagram.

    Typically, you would use a more advanced error handling scheme, but this provides a basic introduction.
11. For better readability, select the **MenuError** connector, and then drag it below the other connectors.
12. Connect the **Goodbye** Play component to the **Voice End** component and label it **End**.
13. Save the application project. Any error indicators (red x's) on the diagram should disappear.
14. Package and deploy the project.
15. When you test this project, remember to speak only the Menu words you added to the grammar.

Currently, the voice diagram should resemble Figure 3.9.

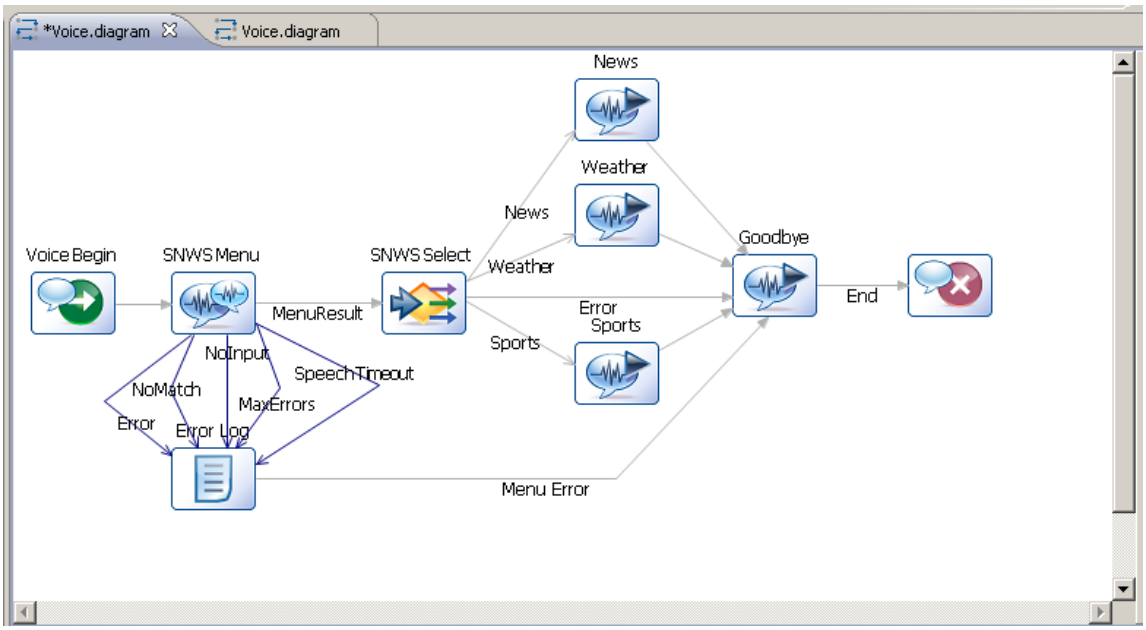


Figure 3.9 - Completed Speech NWS Voice Diagram

Recipe 4: Personalization and Databases

Approximate time: 35 minutes, depending on prior experience

What you will learn with this recipe:

- Obtaining a Caller's Phone Number Automatically
- Passing Variables between Domains Using a Name List
- Using a Database for Persistent Storage
- Using an If Statement
- Using TTS to play the content of a Variable
- Using Built-in grammars

Overview of the Personalization Recipe

With this recipe, you will build a Personalization application project. Personalization means that this application will try to identify the caller and then tailor the subsequent dialog to that particular caller. This is a very simple personalized application, but it provides the framework for a large variety of personalization applications.

In the web world, websites are personalized by requiring a user to log in to the website, thus identifying the user. Once logged in, the user can set various preferences for the site, including the language used, site layout, and much more. When the user returns to that site on subsequent visits and logs in, the user's preferences are stored so that the application can implement them.

The telephone system has a caller ID feature that presents the caller's phone number and/or the caller's name to the called party during the ringing or alerting process. In many automated applications, this caller ID is sufficient to identify the user to the application, though more secure applications can require a PIN or other identification from the caller before accessing sensitive information. Caller ID allows telephone applications to immediately recognize who is calling without making the caller type in logon information.

IVR applications can take advantage of the caller ID capability to simplify and personalize applications. Personalized preferences and interactions such as presentation language, task sequences, reminders, and immediate access to non-sensitive information, can be driven from the caller's ID, which greatly simplifies regular users' experiences.

New caller— For a new caller or someone calling from a new phone, the caller's phone number is not in the database. If the application's database query doesn't return a match for the phone number, the application will assume that the caller is new.

- The system will prompt for a four-digit account number, play it back to the caller, and then store the account number in the database along with the caller's phone number.
- When the system adds the caller's phone and account numbers to the database, it creates a record in the database for that caller. Both should be stored in the same database record.

Existing caller—When the call is received, the Personalization application captures the caller's ID (phone number) and then uses that phone number to query the database. The phone number is the "key" to that user's record, and the account number is just another field in the caller's database record.

- If there is a match for the caller's phone number in the database, the database returns the record.
- The application extracts the account number and therefore does not prompt the caller for it. Instead, the application simply confirms to the caller: "Your account number is XXXX." The TTS engine renders the account number to the caller.
- The caller's record can contain any other personal information that you elect to store in the database.

This sample application project utilizes a database, which is not a standard part of the IC runtime environment. Therefore, this sample recipe will make some assumptions about the database. For runtime when you test your recipe, be sure to use an appropriately configured database.

Creating the Voice Diagram

Tip: For this sample project, to save time, you can copy the sample names (usually in bold font) from this document and paste them into the fields. However, be sure you manually type in any single quotes as Interaction Composer does not recognize the curly font.

1. Create a new application project and name it **Personalization**. For a reminder, see Recipe 1.
2. After creating the Personalization project, create a **Voice_Personalization.diagram** in the **Personalization > applications > voice** folder.

Creating the Call Control Diagram

For the next step, create a **Call_Personalization.diagram** in the application > call folder. For a reminder of the steps, see Recipe 1, “Creating a Call Control Diagram.”

Passing Variables from Call Control to the Voice Diagram

Next, configure the **Call_Personalization.diagram** to obtain the caller’s phone number from the caller ID information. After the call control program has the caller’s phone number, pass the phone number to the Voice diagram for further processing.

To associate the Call Control diagram with the Voice diagram:

1. In the Call_Personalization.diagram, click the **Dialog** icon to select it.
2. In the Properties view > Src field, select the **Voice_Personalization.diagram**. This indicates to the call control diagram which voice diagram to execute after answering the phone.
3. Save the application project. This should clear the red x errors from your call control diagram.

To assign a variable name to the callers’ phone number (ANI):

1. In the Call_Personalization.diagram, click the **Call Begin** icon.
2. In the Properties view > ANI Variable field, enter a variable name, such as **CallerPhoneNo** (no quotes)

   Tip: Copy CallerPhoneNo from this text, because you’ll be pasting it into several fields.

   Note: ANI stands for Automatic Number Identification. It will be the caller’s phone number, which is part of the Caller ID data passed during the ringing process.

   Note: This sample project won’t use the DNIS information in the Caller ID, so you don’t need to assign DNIS a variable name. DNIS stands for Dialed Number Identification Service which lists the phone number that the caller dialed. See the Properties view in Figure 4.1.

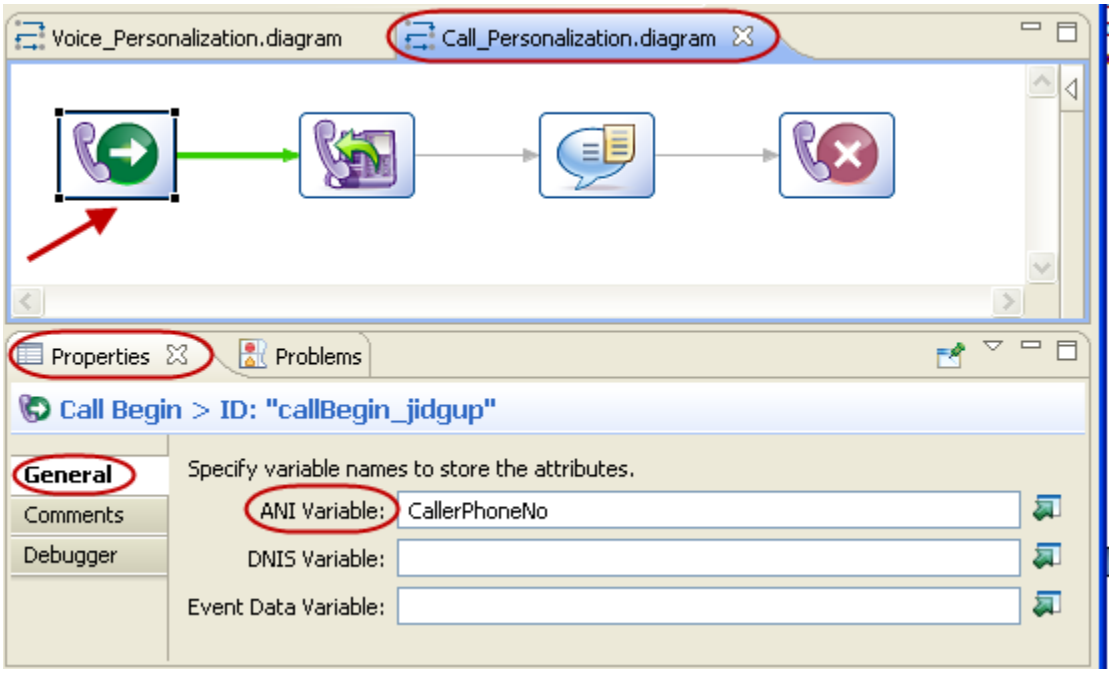


Figure 4.1 - Defining the ANI Variable Name

Next, add the CallerPhoneNo variable to the document's Name List.

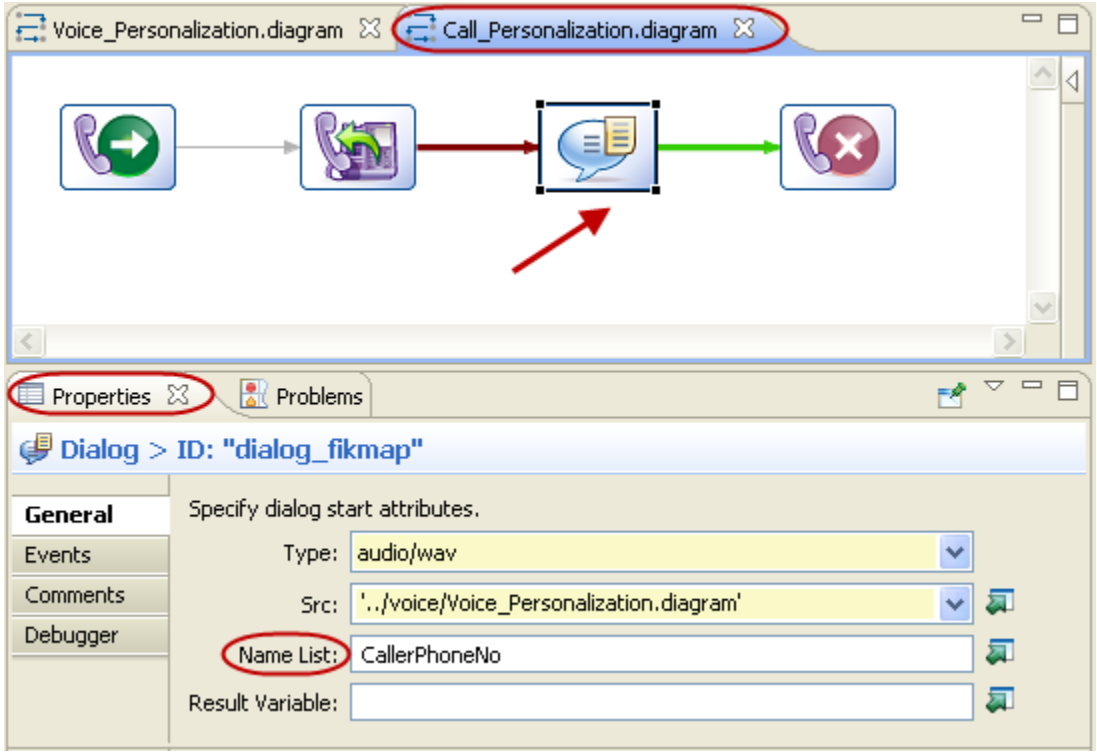


Figure 4.2 - Passing the Caller Phone Number variable

To add the CallerPhoneNo variable to the document's Name List:

1. In the Call_Personalization.diagram, click the **Dialog** icon.
2. In the Properties view > Name List field, type **CallerPhoneNo** as the variable name (no quotes). See Figure 4.2.

   This causes the variable to be passed to the Voice diagram for more processing after the Voice diagram is called from the Call Control diagram. After the Voice diagram has the phone number, it can check for a match in the database.
3. Save the project.

   Note: As an alternate approach, you could design the call flow to do the database check in the Call Control diagram and pass the database results in the Name List. Either approach can work, but this sample project will keep all the database operations in the Voice diagram.

Capturing Name List Variables in the Voice Diagram

To capture the Name List variables:

1. At the top, select the **Voice_Personalization.diagram** tab.
2. Click the **Voice Begin** icon.
3. In the Properties view > General option, click the **Append Row** button on the right.
4. In the Parameter Name column, type **CallerPhoneNo**. (See Figure 4.3)

   CallerPhoneNo is the parameter name referenced in the Dialog component as the variable in the Name List. It names the parameters to be passed from the call diagram to the voice diagram.

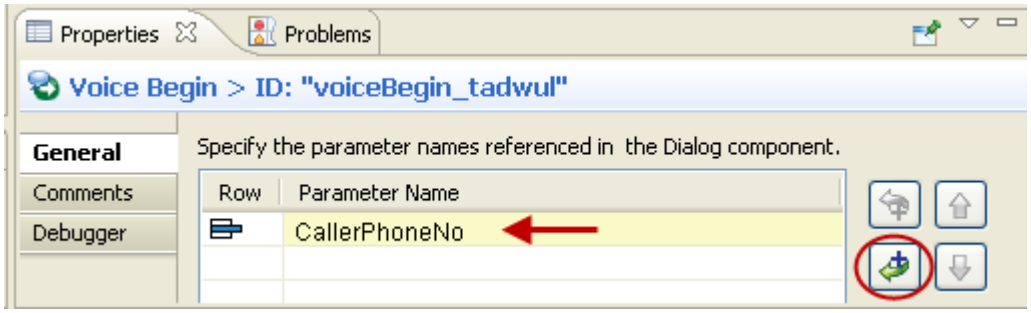


Figure 4.3 - Naming the Passed Variable

5. Save the application project.

Checking the Database

The system now has the caller's phone number. To determine whether this is a new or repeat caller, the system can query the database to see if that phone number is present in the database.

To create the call flow:

1. In the Palette > Services drawer, double-click the **Database Retrieve** component.
2. Label it **RetrievePhoneNo**. See Figure 4.4.
3. In the Properties view > Data Source and SQL fields, see Figure 4.4 **or** enter **your own** database schema.

   Important: Since the detailed database component properties such as the database name and SQL query are dependent on which database you use and how the database schema is structured, this screenshot has sample text that you can modify later when you set up the actual database and know the correct values.

   - Sample Data Source (modify for your own database): **'java:/comp/env/persdb'**
   - Sample SQL (modify for your own): **'select * from persdb where ANI = ?'**
4. In the Properties view > Result Variable field, type a variable name that will carry the result of the database query, for example, **CallerRecord**. It will pass the results of the query to other components.
5. In the Row Count Variable field, type **RowCount**. That puts the number of rows returned from the database query in the RowCount variable. This variable can check whether the query was successful.
6. In the Error Variable field, type **DBError**. That places any error returns in the DBError variable. This variable checks whether any errors occurred on the query.
7. To the right of the SQL Parameter area, click the **Append Row** button.
8. Since the variable name for the SQL Parameter already exists for this sample project, click the ellipsis button (three-dot button) on the right of the Expression field. In the Expression Editor, expand the **Document** variable folder, double-click **CallerPhoneNo**, and then click **OK**.

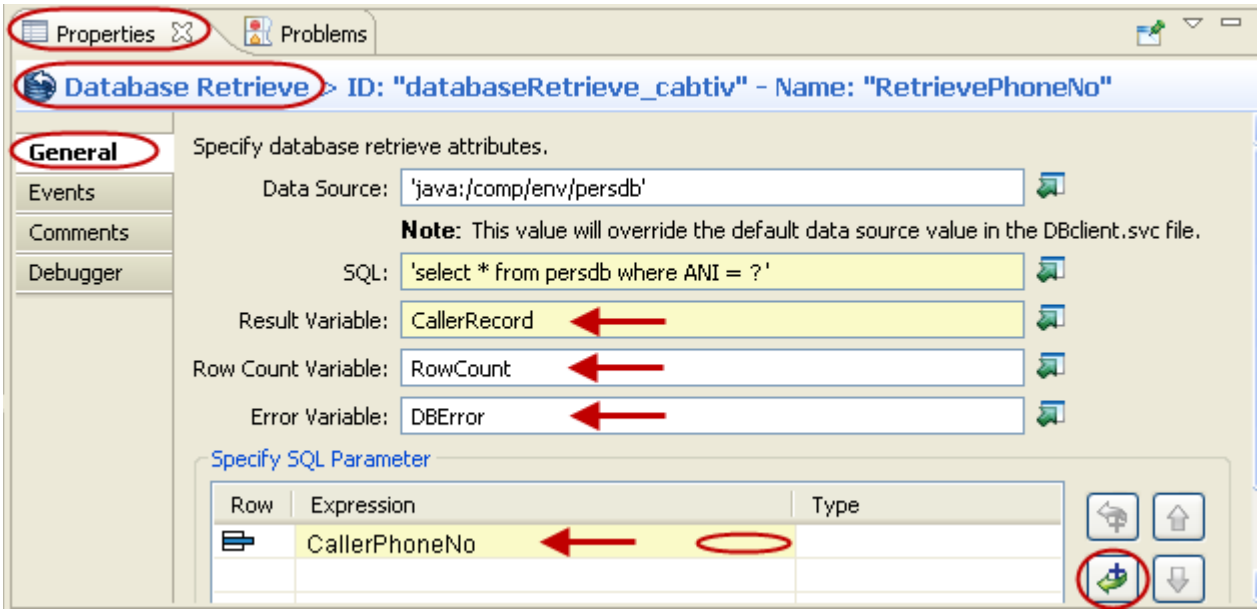


Figure 4.4 - The Database Query and the Database Properties

Using the If Component to Select a Path

To check whether the database query was successful:

1. In the Palette > General drawer, add the **If** component to the right of the Database Retrieve component.
2. Label it **CheckPhoneNo**. See Figure 4.5.
3. Select the **If** component.
4. In the Properties view > Expression column, type **RowCount == 0**.
5. In the Label column, type **NotFound**.
6. At the bottom, in the Else Label field, type **Found**.
7. Specify conditional expression and unique name fields,

   This defines the exit paths from the If component:

   - **Found**—The call flow takes this path if the database query returns a caller's record.
   - **NotFound**—The call flow takes this path if the query returns empty.

   To check whether the length of the data in the CallerRecord variable is zero, the Database Retrieve component provides a RowCount variable that returns the number of rows or database records that the query returns. The ECMAScript expression for the test is **RowCount == 0**. This expression will be true if the CallerRecord is empty.

   If the CallerRecord variable is empty, the RowCount will be zero, and the ECMA expression you typed will be true. If the expression is true, the component will take the exit labeled in the Label field, in this example, **NotFound**.

   If the CallerRecord variable is NOT empty, the call flow takes the Else exit path from the If component. In this example, it is labeled **Found** to indicate that there was something in the record returned from the database. See Figure 4.5.

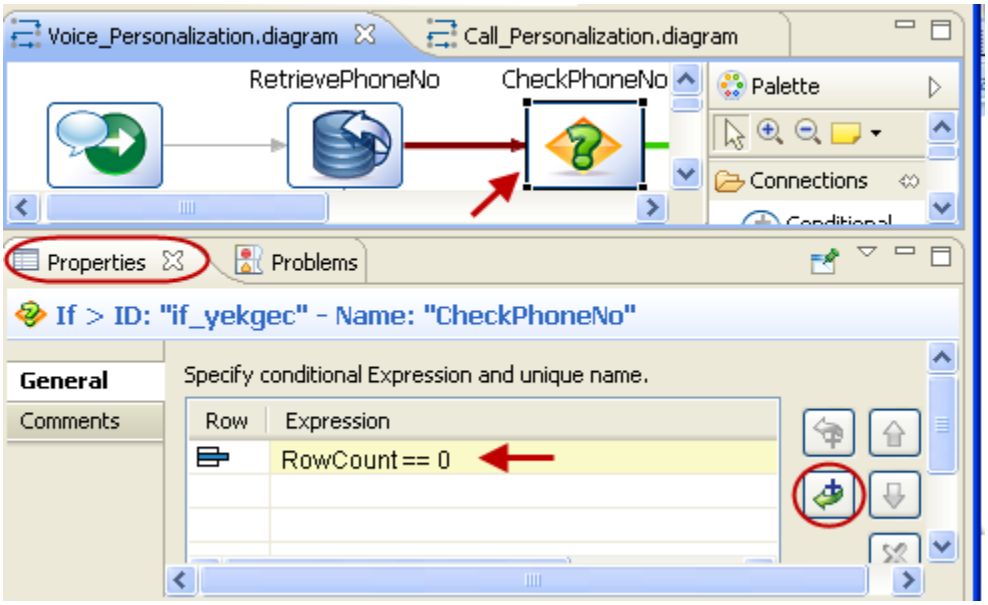


Figure 4.5 - Checking the Database Query Result

Playing the Account Number

This technique is used extensively in many applications, so it is useful to understand the process.

If the caller's phone number is found in the database, the system can use the Database Retrieve component to retrieve the phone number record from the database. In this example, you named this ECMA object **CallRecord**.

The caller's account number should also be stored in that same record. For this example, assume that the database schema placed the caller's account number in the **CallRecord.acctNum** section of the CallRecord object and that it is a text variable. Design this sample call flow to:

- Pass the TTS engine the CallRecord[0].acctNum variable
- Play that account number to the caller.
- Play goodbye and hang up.

1. Add a **Play** component to the right of the CheckPhoneNo If component.
2. Label it **PlayAccountNum**.
3. Select the **Play** component.
4. In the Properties view > General option > Prompt tab, select **Prompt – no text**.
5. In the Type field, select **TTS**.
6. In the Expression field, use single quotes and type 'Welcome back. Your account number is.'
7. Click in the Prompt area, and the text is added to its name.
8. Right-click the new prompt, and select **Insert a new prompt below**.
9. Select **Prompt – no text**.
10. In the Type field, select **TTS**.
11. In the Expression field, type the ECMA name for the account number: **CallRecord[0].acctNum**

    Multiple prompts in a single prompt group will be played sequentially. The first prompt plays the message. The second prompt inserts the caller's account number as a text string. The TTS engine renders both and plays: "Welcome back, your account number is 1234."

    Congratulations! You have just created your first compound prompt! A compound prompt mixes a fixed prompt with a variable prompt to deliver some information to a caller. Now, every time the same caller calls, the application plays their account number.

Connect the components on the voice diagram:

1. Select the connection between the Voice Begin and Voice End components.
2. Click the Voice End's terminating handle, and drag it to the **RetrievePhoneNo** database component.
3. Connect the **RetrievePhoneNo** database component to the **If** component.

   All of these connections are default connections, so you don't have to make any connection type selections.
4. Connect the **If** component to the **Play** component.
5. Select **Create Default Connection** since the Else condition of the If component is the default path. Previously, you labeled the Else path **Found** so the connection is automatically labeled now.
6. On the toolbar, click the **Arrange All** icon. See Figure 4.6.

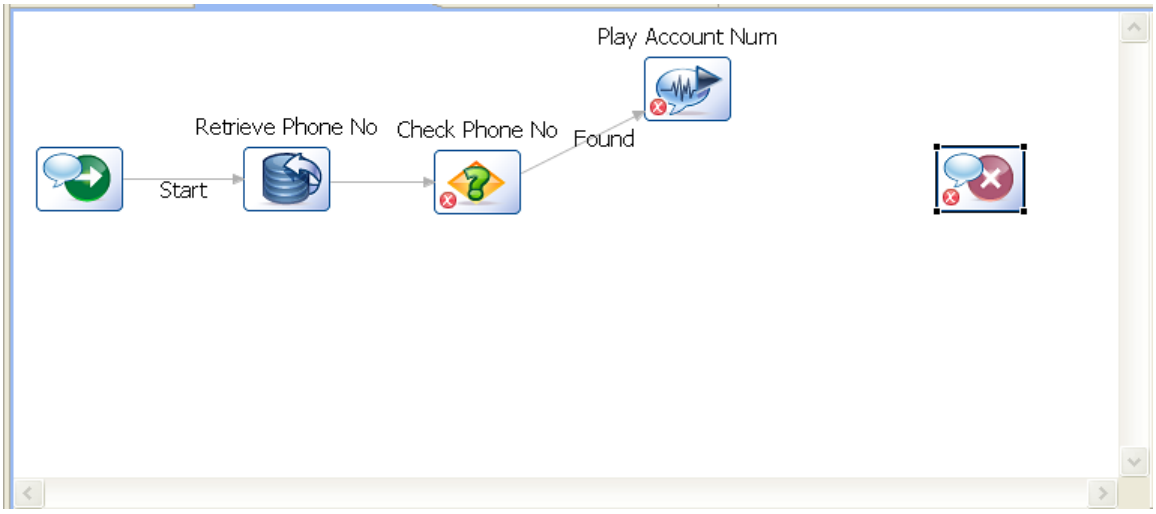


Figure 4.6 - Connecting the Found Path

Collecting a New Account Number

If a caller's phone number is not found in the database, prompt the caller to enter it using the keypad on their phone. After the system collects the phone number, code the call flow to save the account number with the phone number in a new record in the database.

To capture the caller's account number:

1. From the Voice drawer, add a **Collect** component to the right and below the If component.
2. Label the component **GetAcctNo**.
3. Select the **GetAcctNo** component.
4. In the **Properties** view > **General** option > **Prompt** tab, select **Prompt - no text**.
5. In the Type field, select **TTS**.
6. In the Expression field, use single quotes and type: 'Welcome new user. Please enter your four-digit account number.' See Figure 4.7.

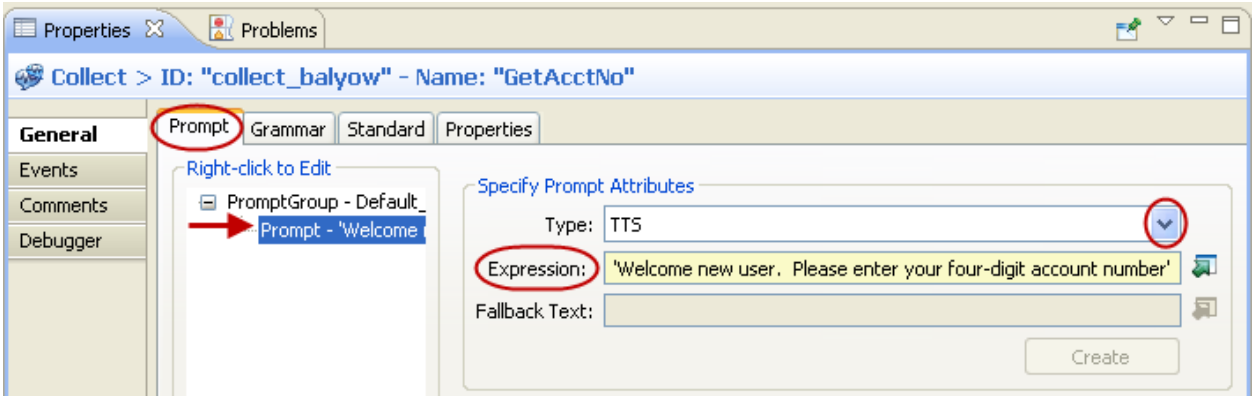


Figure 4.7 - Collect Prompt Properties

7. Select the **Properties** view > **General** option > **Grammar** tab.
8. In the Mode field, select **'dtmf'**.
9. Accept the default weight, '1.0'.
10. In the URL field, specify the URL as: **'builtin:dtmf/digits?length=4'**

    This is a built-in grammar defined in VoiceXML. When you set the length to **4**, the user must key in exactly four DTMF digits. As soon as the user types in the fourth digit, the call flow will exit the Collect component in the default path. See Figure 4.8.

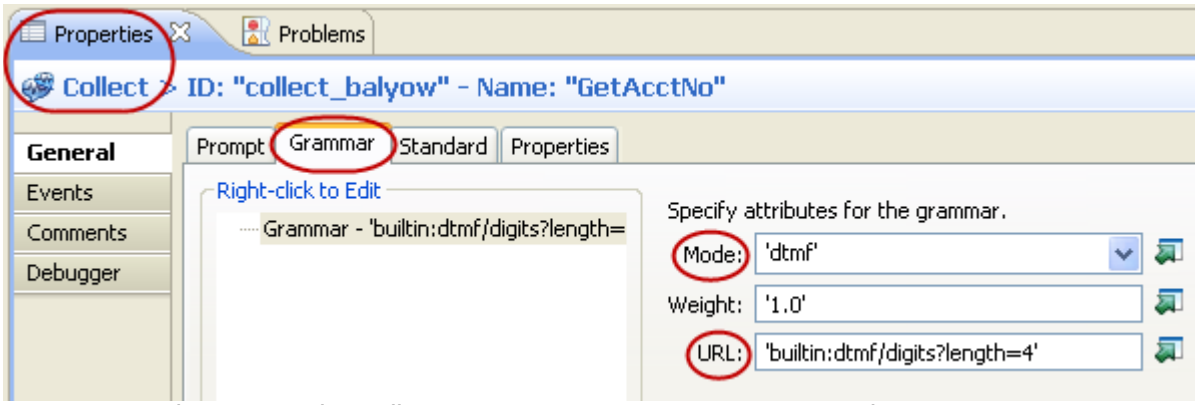


Figure 4.8 - The Collect component DTMF Grammar properties

11. Select the **Properties** view > **General** option > **Standard** tab.
12. In the Slot Name field, type **'MEANING'** including the quotes.
13. In the Result Variable, type **New_Acct_No**. See Figure 4.9.
14. In the Input Mode field, select **'dtmf'**.

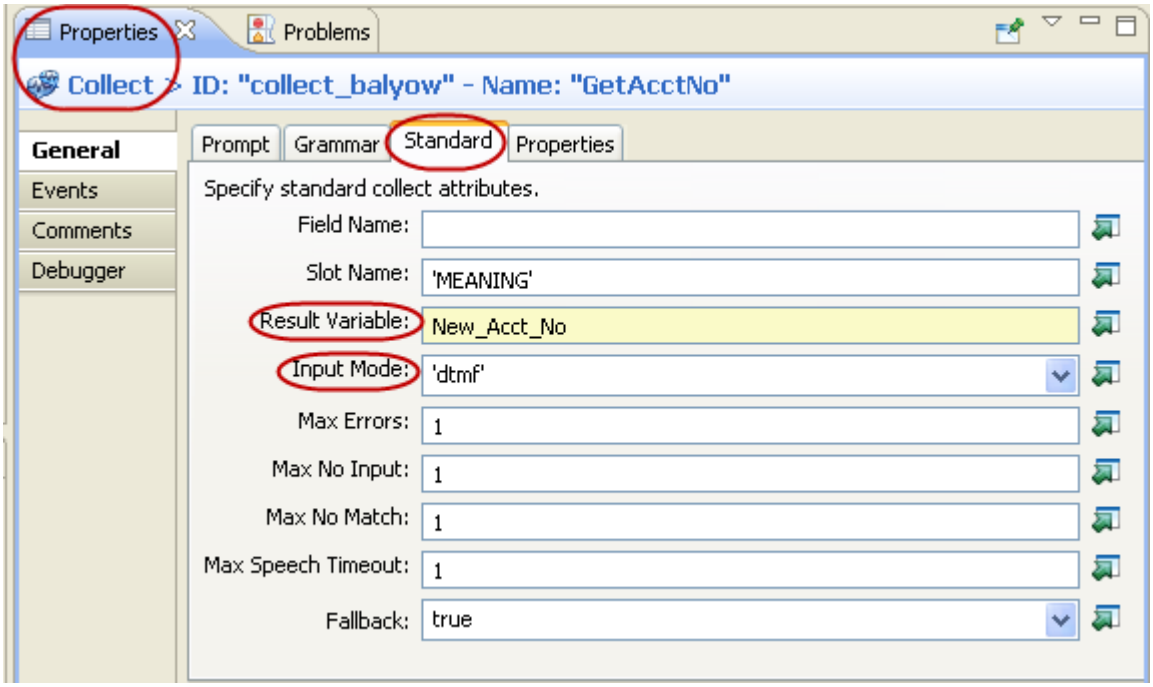


Figure 4.9 - Naming the Collect Component Result Variable

15. Because this is just a sample recipe for Personalization, in the **Properties** view > **Events** option, accept the default error events as **Ignore**.

    This is not good VUI practice, as it routes all of the error paths out the same default path as the success path. Future recipes will explain error handling options and procedures.

16. Connect all the components.

Storing the Account Number and Phone Number

To store the caller's account number in the personalization database along with the caller's phone number:

1. From the Services drawer, add a **Database Edit** component to the left of the GetAcctNo component.
2. Label it **StorePhoneAndAcctNo**.
3. Select the **Database Edit** component. See Figure 4.10.

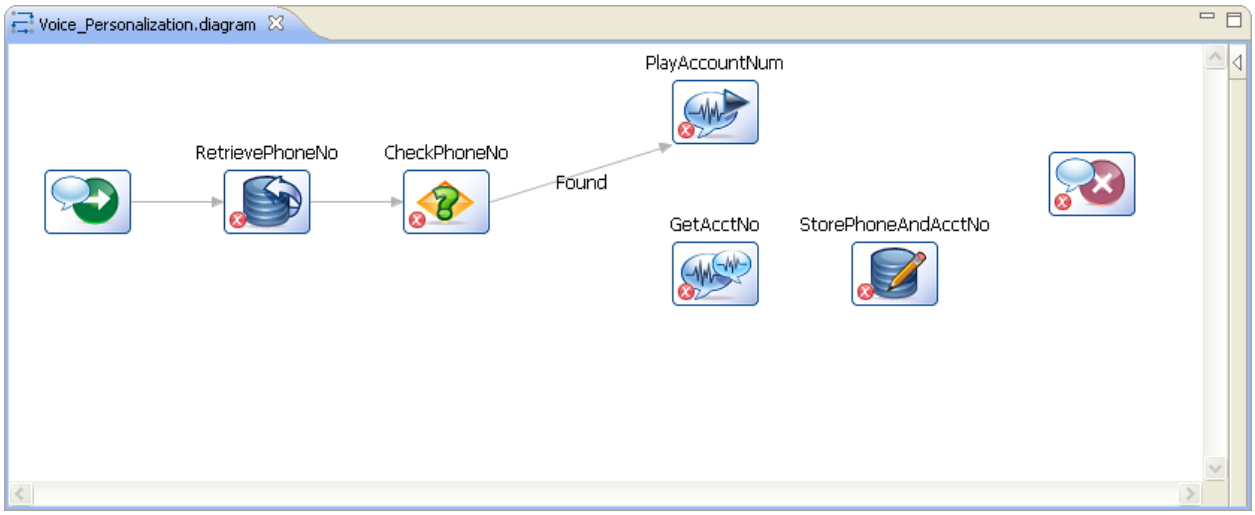


Figure 4.10 - Get Account No. and Database Edit components

4. The Properties view > Data Source field defines the database that the system will access. The exact content will depend on your database schema. Here is an example:

   'java:/comp/env/persdb'

5. The SQL field must contain the query to be executed, and it needs to reference the two parameters in the SQL INSERT statement. Again, the exact content will depend on your database schema. Here is an example (enclosed in single quotes):

   'INSERT INTO perstable (?, ?) VALUES'

   Note: The SQL statement can't use any variables unless they are defined in the SQL Parameter area.

6. To the right of the SQL Parameter area, click the **Append Row** button twice.
7. Click in the first Expression row, and then click the ellipsis button. The Expression Editor opens.
8. In the left pane, expand the Document Variables folder and double-click the **CallerPhoneNo** variable. The name appears in the Edit expression area at the bottom.
9. Click **OK**. The variable is added to the Expression field.
10. Click in the second Expression row.
11. Enter the value **New_Acct_No.result**
12. Click **OK**.

    The row values of the SQL Parameter list are assigned to the corresponding values in the SQL query on a top-down, left-right basis. The parameter in the first row (i.e. topmost) of the SQL parameter list is assigned to the left most (i.e. first question mark) variable in the SQL query. See Figure 4.11.

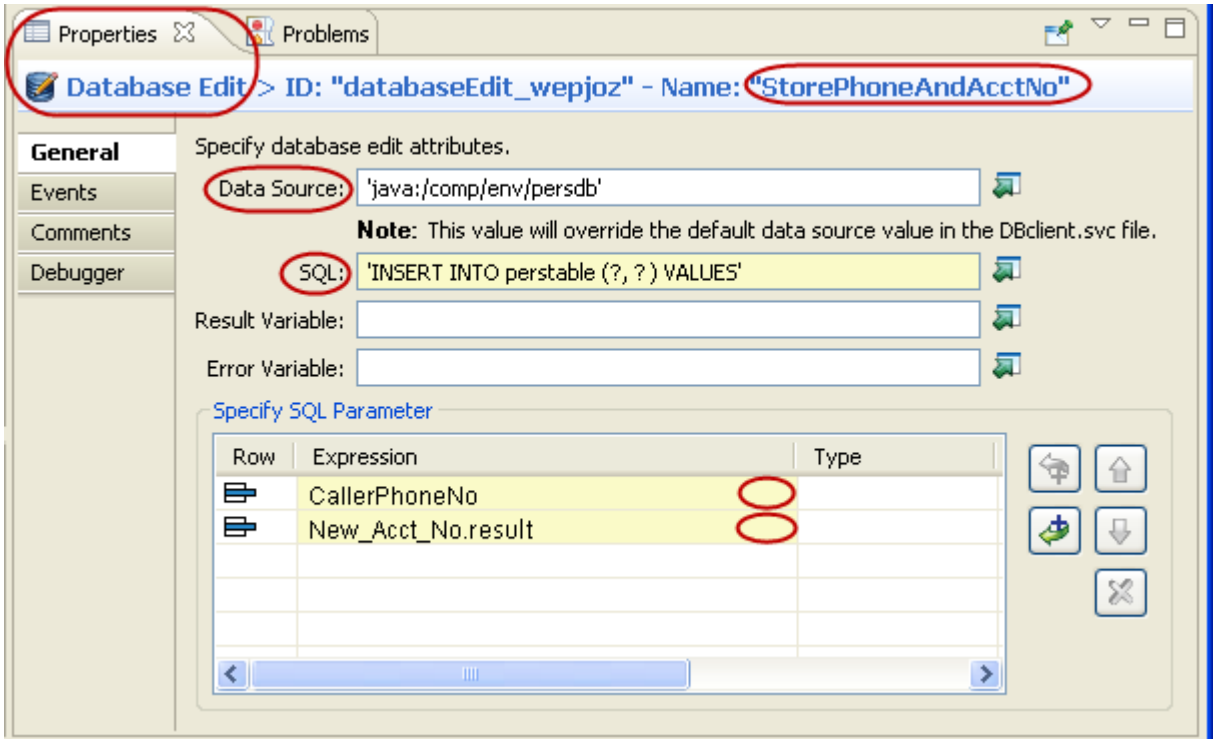


Figure 4.11 - Database Update properties

Saying Goodbye

1. Connect the **If** component to the **GetAccountNo** Collect component, and click the **NotFound** pop-up as the exit path.
2. Connect the **Collect** component to the **StorePhoneAndAcctNo** Database Edit component. No label is necessary.
3. From the Voice drawer, add a **Play** component and label it **Goodbye**.
4. Modify **Prompt – no text** to a **TTS** prompt: 'Thank you, and goodbye.'
5. Route the **PlayAccountNum** and the **StorePhoneAndAccountNo** legs of the call flow to the **Goodbye** component. No labels are necessary.

Remaining Diagram Errors

To get rid of an error condition on the database query component:

1. Select the **RetrievePhoneNo** component, and go to **Properties** view > **Events** option.
2. In the Error field, select **Inline**.
3. Connect the **RetrievePhoneNo** component to **Goodbye** component. Select the **Error** pop-up.

   If an error occurs on the first query, the system will play the Goodbye prompt and end the call. This isn't good UI practice for error recovery, but this recipe focuses on personalization.
4. Connect the **Goodbye** component to the Voice End component.
5. Arrange the arrows to tidy up your diagram.
6. Check the **Problems** view and correct any Errors. If you followed the steps, you shouldn't have any.

You have completed the Voice diagram for the Personalization application. See Figure 4.12.

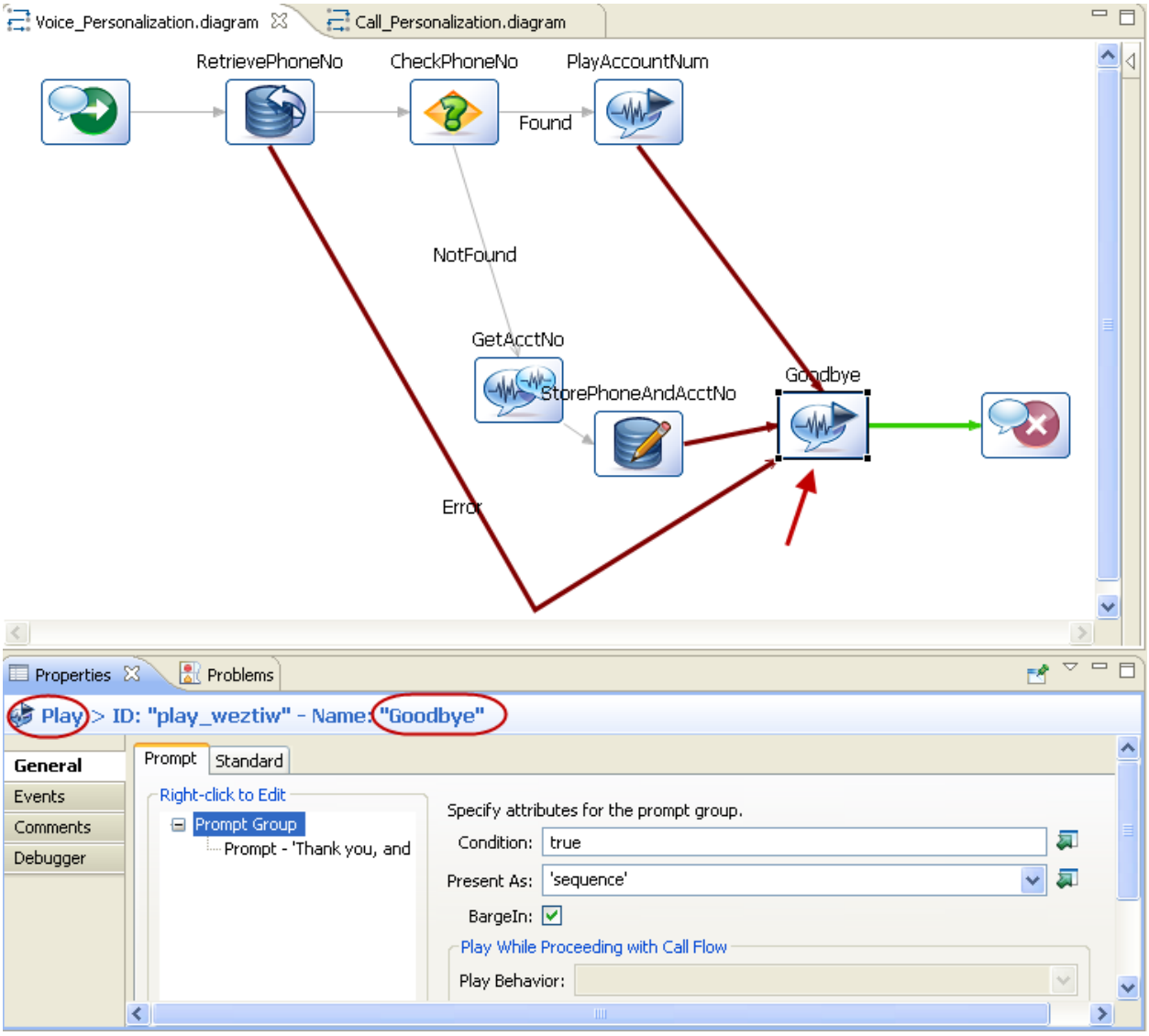


Figure 4.12 - Final Personalization Recipe

Remember, this application utilizes a database, which is not a standard part of the IC runtime environment. For runtime when you test your recipe, be sure that an appropriately configured database is present.

To test the finished application, call the system from two different phone numbers and give two different account numbers. When you call back from either one of the phones, you'll hear the account number associated with that specific phone.

Congratulations again! With the personalization recipe, you have developed the framework for many powerful applications.

Recipe 5: Accessing Web Services

Approximate time: 45 minutes, depending on prior experience

What you will learn with this recipe:

- How to use Web Services
- What a WSDL is
- How to create a web services library subroutine
- How to pass variables to/from a subroutine
- Finding the Java methods and attributes
- How to speak the data obtained from a web service
- How to handle errors

Requirements:

- Correct version of the **Java SE JDK (Standard Edition)** development kit on your development system. The web wizard will remind you which JDK SE version must be used with your IC version.
- Internet access on your system, since you will be using a real weather web service to build your application project.

What the Web Services Recipe Does

This recipe builds an application that will report the current weather at a user's location. When the system answers the call:

- The application will prompt the user for a zip code to determine their location.
- The application then calls a web service to provide a current weather report for that zip code.
- The weather report text is extracted from the web service weather object and played to the user through text-to-speech.

The Web Services Wizard

This recipe explains how to use the ICS web services wizard to develop a set of library routines to access a web-based current-weather report and forecast for any U.S. zip code. You must enter the URL of the web service WSDL document, and the wizard completes the request.

A web service uses standard web protocols to provide dynamic information over the web. A web service defines the information that it can provide using a web document written in a dialect of XML called WSDL (Web Services Definition Language). Typically, the WSDL document is available on the web, so it can be downloaded by any process wanting to access that web service. The WSDL document describes the details of the service interface and is designed to allow an automated process to discover the access methods for the service and create the processes required to access those services.

This is also what the Interaction Composer web services wizard does. If you point the web services wizard to a WSDL doc on the web, it will access the document and use it to create all the methods required to access that web service. The wizard puts all of those processes in an IC library project so that they can easily be called by your application.

Using the Web Services Wizard

Tip: Close any open diagrams and collapse the folders in the Package Explorer.

To use the Web Service Library wizard to build the weather service access:

1. From the Interaction Composer menu bar, select **File > New > ICS Web Service Library**.

   The ICS Web Service Library wizard opens (see Figure 5.1).

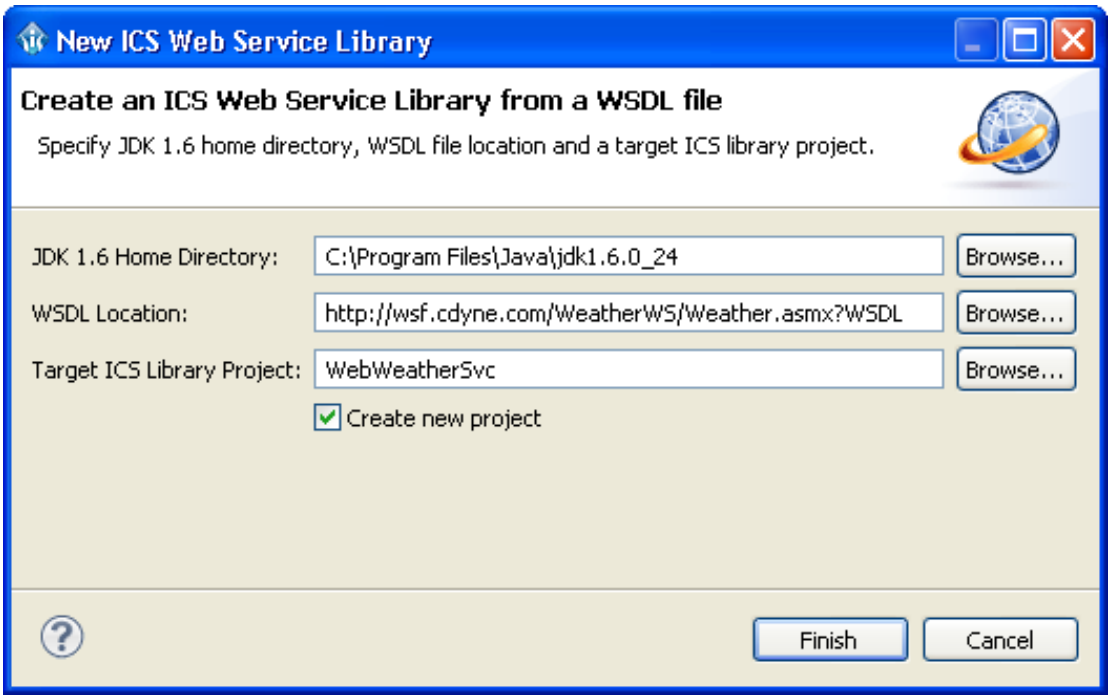


Figure 5.1- Web Services Wizard

2. To point the wizard to the location of your JDK main folder, click **Browse** next to the JDK x.x Home Directory field and select your folder location, for example, **C:\Program Files\Java\jdk1.6.0_24**.
3. In the WSDL Location field, enter the URL of a location for a weather service WSDL document on the web, so that the system will know how to retrieve the WSDL doc.

   A typical weather web service is provided by the web services company Cdyne. Please note that Convergys has no control over this web site and WSDL.

   In the wizard's WSDL Location field, enter this URL, which is the location of their weather service WSDL document:

   http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL

   A test web page that uses the Cdyne weather web service is located at: http://ws.cdyne.com/WeatherWS/Weather.asmx?op=GetCityWeatherByZIP

   You can use that browser page utility to view the XML WSDL document that the web service returns.
4. In the Target ICS Library Project field, enter a name for the weather web service library project. In this sample library project, enter **WebWeatherSvc**.
5. At the bottom right, click **Finish**.

   The wizard creates the library project WebWeatherSvc. The wizard will access the WSDL over the web and place the weather services it created in the new library project folders.

It may take a few seconds to create the new library. When it finishes, the folders are listed in the Package Explorer. See Figure 5.2.

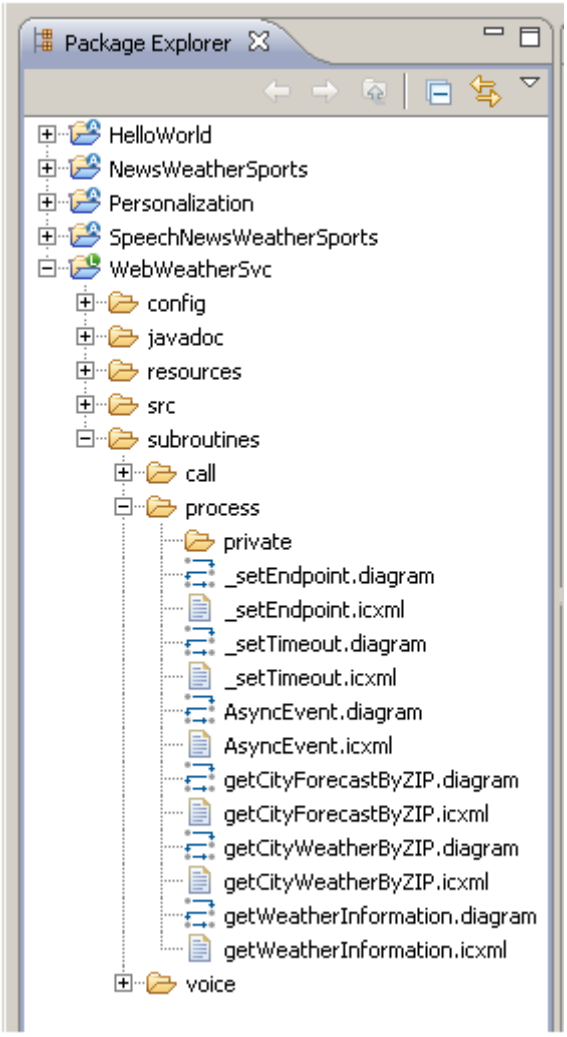


Figure 5.2 - Web Weather Service Folders

Based on the WSDL provided in the wizard, the wizard created several subroutines that can access the weather service. In the Package Explorer, expand the **subroutines > process** folder to see the names of the diagrams and to gain an idea of what the subroutines will do.

6. To display the weather subroutine process diagram that the web services wizard created, double-click the **getCityWeatherbyZIP.diagram** file.
7. In the diagram window, select the **Script** component (yellow scroll icon).
8. In the **Properties** view > **General** option, notice the contents of the script. See Figure 5.3.

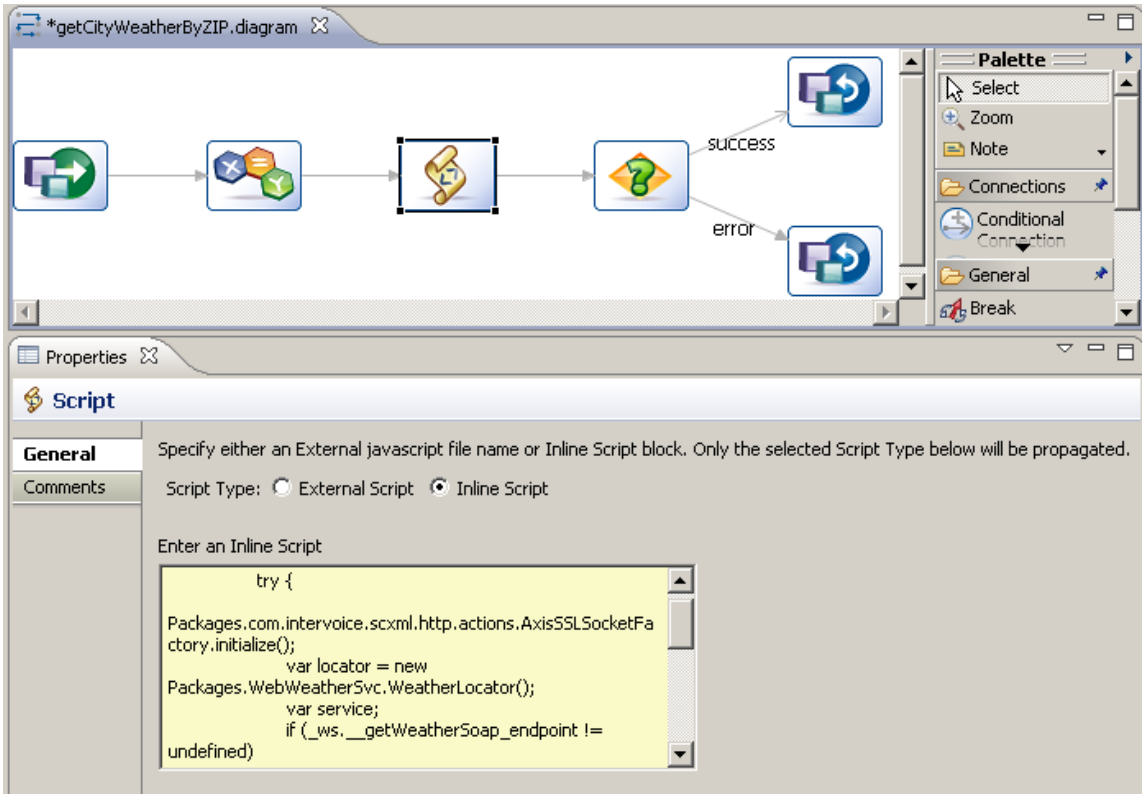


Figure 5.3 - Process Diagram for the Web Weather subroutine

In the diagram, notice that the wizard generated two outcomes for the process subroutine

- success path
- error path

9. To use this subroutine, expand the **WebWeatherSvc > subroutines > voice** folder and open the Voice diagram.
10. On the Voice.diagram, add a **Subroutine Call** component.
11. When you connect that Subroutine Call component to subsequent components in the main diagram, you will see two choices for the connection – a success connection and an error connection. IC automatically configures the exit paths of subroutine components to match the number of outcome paths in the subroutine.

The Properties view of the subroutine displays the auto-generated JavaScript code that the wizard created to access the various Java classes provided in the weather service. To see the actual Java code that the wizard created for the weather services:

1. In the Package Explorer, navigate to **WebWeatherSvc** project > **src** folder > **WebWeatherSvc** folder.
2. Double-click each **.java** source filename to view the Java source for the various methods.

Tip: In a live application, there may be delays before the weather web service returns the weather report data. You may have to modify the web services' Timeout parameter from the default to deal with longer-than-normal delays. You can also redirect the location of the web service using the Endpoint method. To change the default values of these parameters in the web service when the default values are not appropriate for your application, insert calls to the **setEndpoint** and/or **setTimeout** methods before you call the getWeatherByZIP method in your main diagram.

Building the Weather VUI

Now, the weather-getting subroutines are in place. To create the main application project:

1. From the Interaction Composer menu bar, select **File > New > ICS Application Project** and name the project **WebWeather**.

   Do **NOT** click Finish yet.

2. Click **Next** to access the Library References wizard. This allows you to link the WebWeatherSvc Library to the main application. See Figure 5.4.

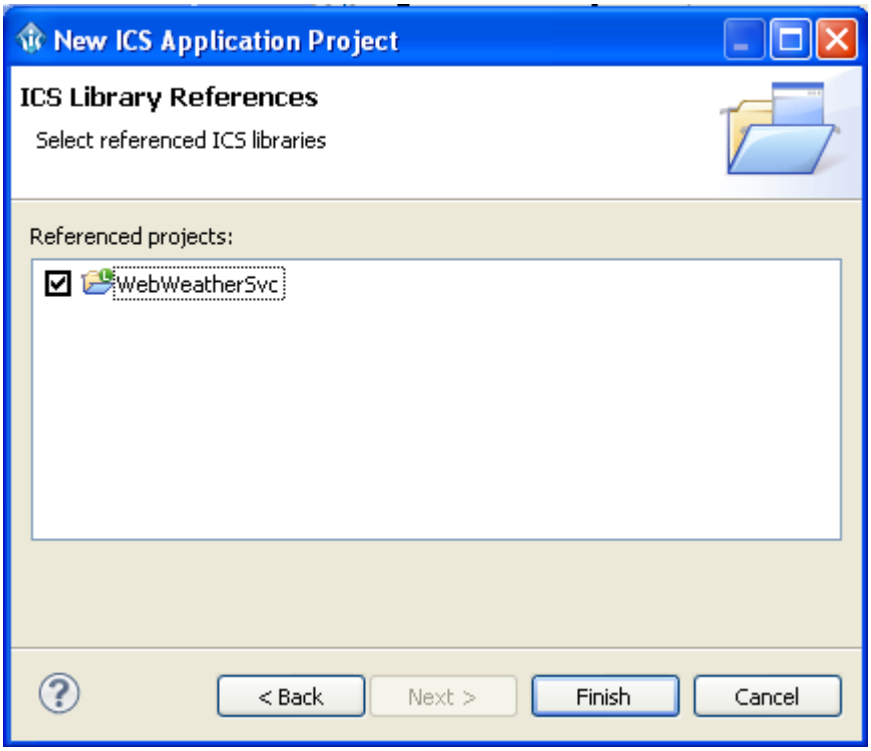


Figure 5.4 - ICS Project Library Reference Wizard

3. Check the box next to WebWeatherSvc to indicate that this library project is to be associated with the WebWeather application project.

4. Click **Finish**. This ensures that when you package the WebWeather application into a WAR, the WebWeatherSvc library will be included in the WAR package.

   This wizard creates a standard IC project where you will build the VUI portions of the weather application. It prompts the caller for their zip code and then calls the WebWeatherSvc library to get the live weather report for that zip-code area.

   The following steps for collecting a caller's zip code are broad guidelines. For detailed steps, see the DTMF collection example in the Personalization recipe.

5. In the WebWeather application project, create a **Voice_WebWeather.diagram**. (See Recipe 1)

6. Add a **Collect** component that will collect the caller's zip-code, and label it **GetZipcode**. Connect the **VoiceBegin** component to the **GetZipcode** component.

7. In a single Prompt Group, add two TTS prompts:
   - Text for first prompt: ‘Welcome to the weather application.’ See Figure 5.5.
   - Second: ‘Please enter the zip-code of the area for which you want the current weather report.’
8. Select the **Grammar** tab. As in the Personalization recipe, the grammar attributes are:
   - Mode field— select **‘dtmf’**
   - Weight—**‘1.0’**
   - URL—**'builtin:dtmf/digits?length=5’**

   Note: The digit length is set to **5** for a zip-code, instead of 4 as in the Personalization recipe.
9. Select the **Standard** tab. In the Result Variable field, type **Zipcode**.
10. In the Input Mode field, select **‘dtmf’**.
11. In the Slot Name field, enter **‘MEANING’**.
12. In Events option > Error, select **Inline**.

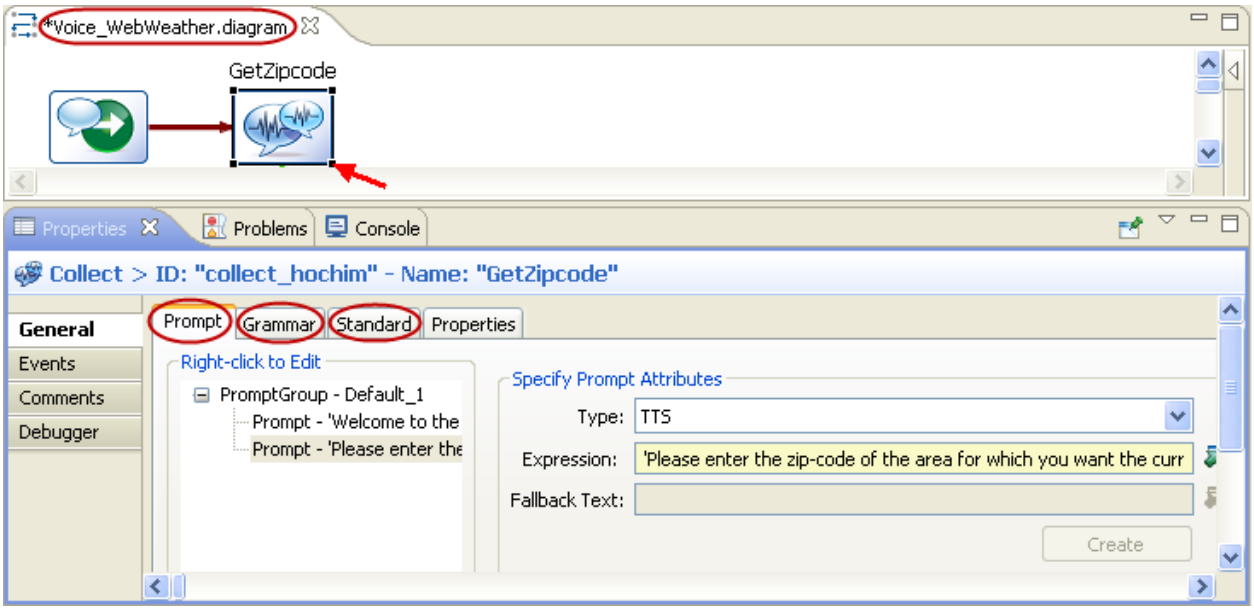


Figure 5.5 - WebWeather Application

Calling the Weather Subroutine

To call the library subroutine to get the current weather report:

1. In the Palette, open the **General** drawer.
2. Add a **Subroutine Call** component to the right of the GetZipcode component on the diagram, and label it **GetWeather**.
3. Select the **GetWeather** Subroutine Call component.
4. In the **Properties** view > **General** option, click the entry selector icon to the right of the **Subroutine Name** field (white square with a green arrow pointing to the upper right). The Subroutine Entry Selector window opens.
5. Expand the **WebWeatherServices > subroutines > process** folders. See Figure 5.6.

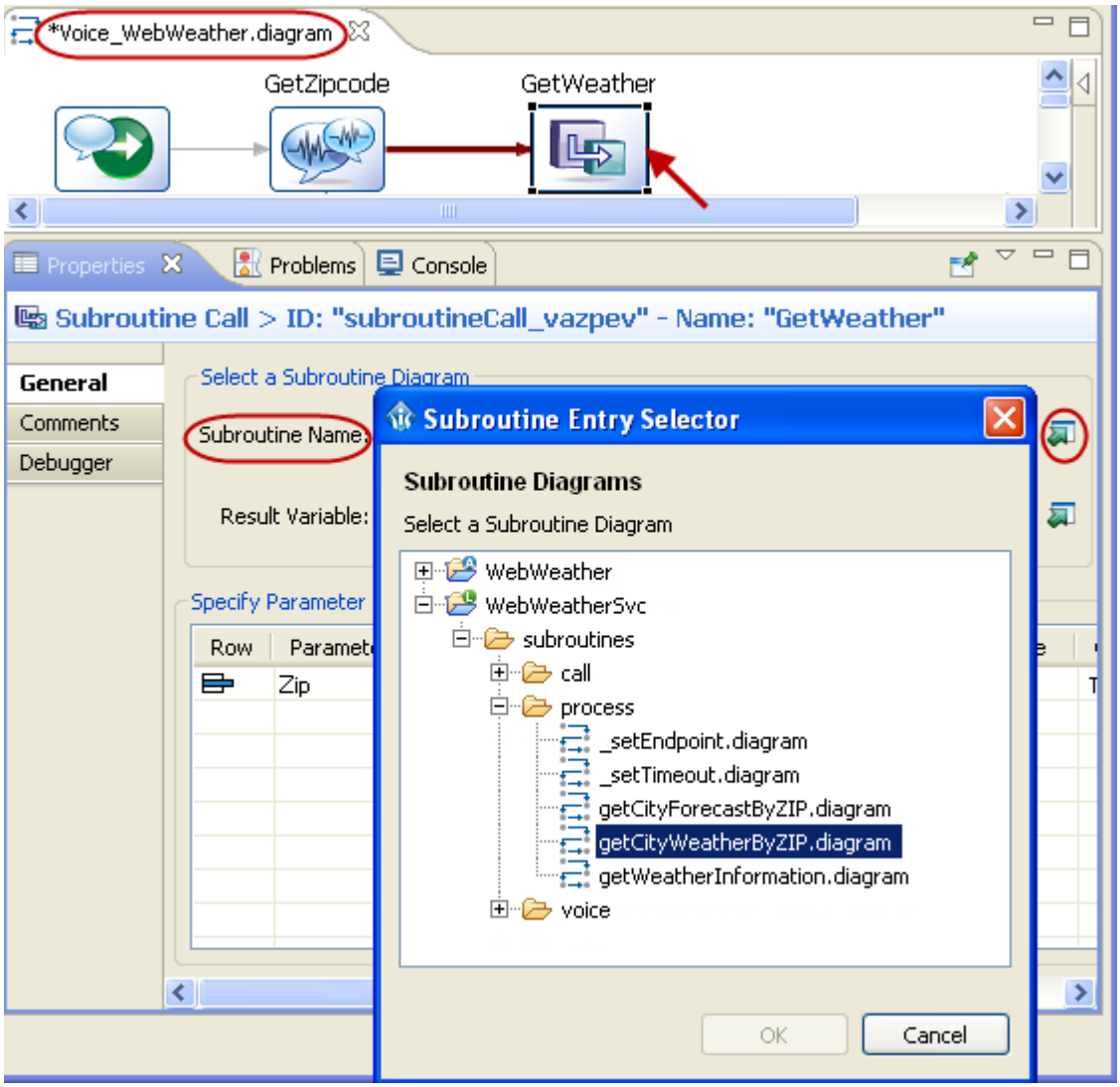


Figure 5.6 - Selecting the Weather Subroutine

This list of subroutines is essentially the same list as the diagrams in the Package Explorer earlier.

6. For this sample application project that reports the current weather, select the **getCityWeatherByZip.diagram** subroutine and click **OK**.

   IC does the following:

   - Populates the ZIP Description field
   - Populates the Parameter Name column with the parameter **ZIP**. IC will try to populate the passed subroutine parameters whenever it discovers them.

7. In the Subroutine Call's Result Variable field, type **CurrentWeather**. This variable holds the weather data that comes from the web service.
8. In the Value column of the row for Zip, enter the value **Zipcode.result**
9. Save the project.
10. To the right of the Result Variable field, click the selector icon.
11. Confirm that you have defined your subroutine input and output variables:

    a. Expand the **Document Variables** folder. You should see two document variables - CurrentWeather and Zipcode. (If CurrentWeather is not there, go back and save the project.)

    b. On the Variable Selector window, click **Cancel**.

11. Connect the **GetZipcode** component to the **GetWeather** component. This is a default connection.

    When the weather subroutine is called, it will access the web service and put all the weather data it receives into the Java object that you named **CurrentWeather**.

12. Save your project.

Finding the Java Methods for the Weather Object

Now, you must identify the specific pieces of weather data that will be contained in the CurrentWeather Java object, so that you can construct the weather report prompt. You will need the names of the Java methods that extract each piece of weather data from the object. You will use those names to generate the spoken weather report for the caller.

To find the callable names of the weather methods in the CurrentWeather object, look at the Javadoc that the web services wizard created in the weather service library:

1. In the Voice_WebWeather.diagram, select the **Get Weather** Subroutine Call component.
2. In the Properties view, look at the Subroutine Diagram's Description field. It should be similar to:

   Method description for getCityWeatherByZIP, **Return type** is WebWeatherSvc**.WeatherReturn**

   This means that the name of the WebWeatherSvc Java class that describes the methods you'll need is called the **WeatherReturn** class.
3. In the Package Explorer, expand the **WebWeatherSvc > javadoc** folder.
4. Right-click **index.html**, and select **Open With > Web Browser**. This opens a browser in the Eclipse Canvas view. The left pane of this Javadoc displays the **All Classes** navigation page.
5. In the All Classes list, click **WeatherReturn**.

   The WeatherReturn class opens with all of that Java objects' attributes and access methods.

   Here is the brief list of the weather attributes contained in the weather object taken from the WeatherReturn Javadoc:

   WeatherReturn Javadoc

   public WeatherReturn(boolean success,
   java.lang.String responseText,
   java.lang.String state,
   java.lang.String city,
   java.lang.String weatherStationCity,
   short weatherID,
   java.lang.String description,
   java.lang.String temperature,
   java.lang.String relativeHumidity,
   java.lang.String wind,
   java.lang.String pressure,
   java.lang.String visibility,
   java.lang.String windChill,
   java.lang.String remarks)

   Figure 5.7 - Java Weather Object Attributes

   The names of methods used to extract these attributes from the Java object are also listed in the Javadoc. For example, examining the Javadoc, the name of the method for obtaining the current temperature is **getTemperature**. If you want to play a prompt of the current temperature, extract the text string containing the temperature using the call **CurrentWeather.getTemperature()**. The open/closed parenthesis indicates the JavaScript method call.

   Tip: In IC, after you open a specific document in a browser, IC will remember that action. Above, you selected **index.html**, and then **Open With > Web Browser**. The next time you open that document, you won't need to navigate again. Just double-click the document. If you double-click an HTML document to open it for the first time, IC will open the document in a text window, displaying the HTML source. Opening HTML docs as source is the default behavior for IC.

Designing the Weather Report Prompt

Before you code the weather report prompt, the best practice is to design the prompt structure. That helps the prompt-coding to go faster.

Here is a sample weather report that can be played to a caller:

“The current weather in Dallas is partly sunny, and the temperature is 75 degrees. Thanks for calling.”

To make this sample recipe easier, use only three weather attributes:

- weather city (Dallas)
- weather description (partly sunny)
- temperature

The weather report prompt will have six sequenced TTS prompts:

- ‘The current weather in’
- CurrentWeather.getCity() - This the JavaScript call for the city name.
- ‘is’
- CurrentWeather.getDescription() - The JavaScript call for the weather description
- ‘and the temperature is’
- CurrentWeather.getTemperature() – The JavaScript call for the temperature
- ‘degrees. Thanks for calling.’

  Note: The text prompts require single quotes. The JavaScript does not.

Playing the Weather Report

To construct the weather report prompt:

1. In the Voice_WebWeather.diagram, add a **Play** component to the diagram, and name it **PlayWeatherReport**.
2. In the Prompt Group, add six additional the prompts for the multi-part weather report. See Figure 5.8.
3. For Type for each, select **TTS**. The text prompt expressions require single quotes. The JavaScript does not. (You can copy below for each prompt. However, manually type in the single quotes.)

   'The current weather in'
   CurrentWeather.getCity() - This the JavaScript call for the city name.
   'is'
   CurrentWeather.getDescription() - The JavaScript call for the weather description.
   'and the temperature is'
   CurrentWeather.getTemperature() – The JavaScript call for the temperature.
   'degrees. Thanks for calling.'

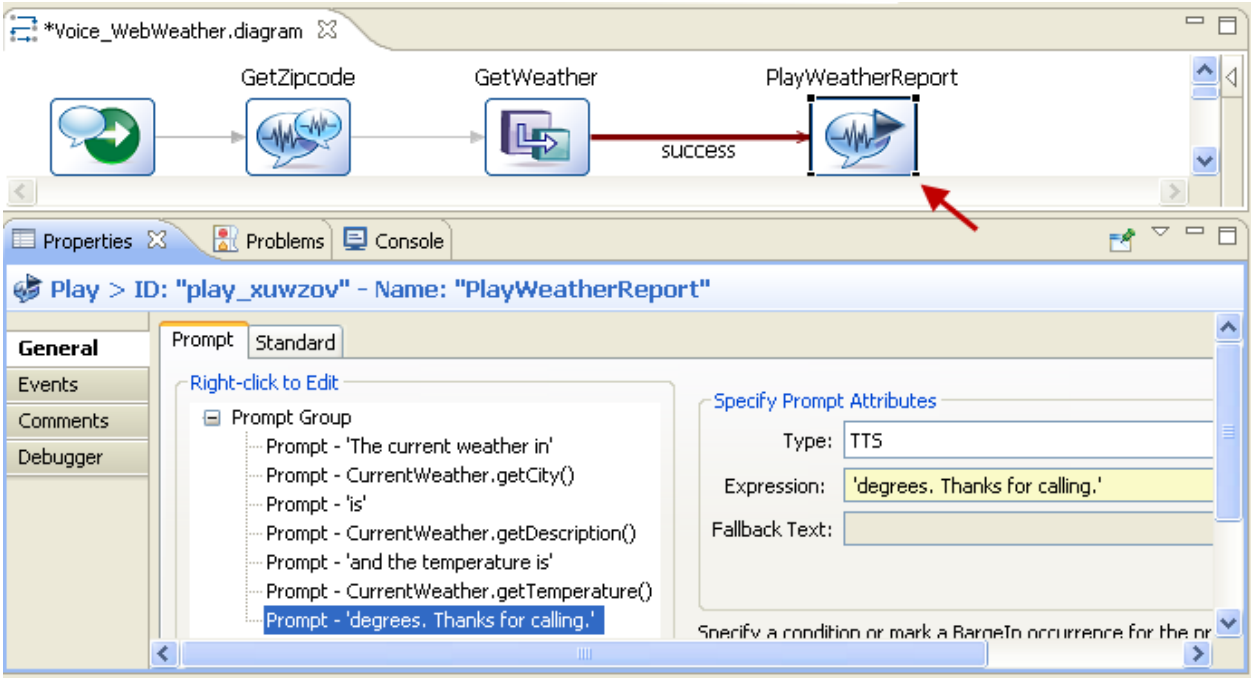


Figure 5.8 - Playing the Weather Report

4. Connect the Get Weather component to the Play Weather Report component.
5. For the connection option, click **success** since the Play Weather Report component is the successful weather report path. On the next page, you'll build an Error path. See Figure 5.8.

Error Handling

The following types of errors can occur:

- Caller doesn't enter a zip code
- Caller enters an invalid code.
- Web service fails, either because the service is down or the connection to the web is not working.

For this sample project, you will design the error leg and two error prompts:

- Zipcode, explaining that the caller has entered an invalid zip code, or didn't enter anything at all.
- web service fails

1. In the diagram under the GetZipcode component, add a **Play** component and label it **ZipcodeError**.
2. In the Properties view, select **Prompt – no text**.
3. In the Type field, select **TTS**.
4. In the Expression field, type '**The zip code you entered is invalid.'**
5. Open the GetZipcode Collect component. On the left side, select the **Events** option.
6. In the event handler's Error field, set property from Ignore to **Inline**. This will configure the GetZipcode component to have a second exit path for all errors.
7. Connect the **GetZipcode** component to the **ZipcodeError** component, and select the **Error** connection property that displays.

   Note: In a production application, the application would probably allow the caller try again for a valid zip code, but in this simple recipe, the caller only gets one chance.
8. Add another **Play** component under the Play Weather Report component, and label it **WebWeatherSvcError**.
9. Connect the **GetWeather** component to the **WebWeatherSvcError** component, and select the **error** connection option.
10. Set the WebWeatherSvcError TTS prompt to play 'The Weather Service is not currently available. Call back later.'
11. In the **WebWeather > call** folder, create a new **Call_WebWeather.diagram** (see Recipe 1).
12. Set the **Call_WebWeather.diagram** to the **Voice_WebWeather.diagram**.

To complete the diagram:

1. Connect the **ZipcodeError** Play component to the **Voice End** component.
2. Select **Create Event Connection** and then select the **Error** popup.
3. Connect the **WebWeatherSvcError** Play component to the **Voice End** component and label it **WebError**.
4. Connect the **PlayWeatherReport** Play component to the **Voice End** component and label it **Success**.
5. Save your project. All red x's should be cleared.

   Figure 5.9 shows the completed **Voice_WebWeather** diagram.

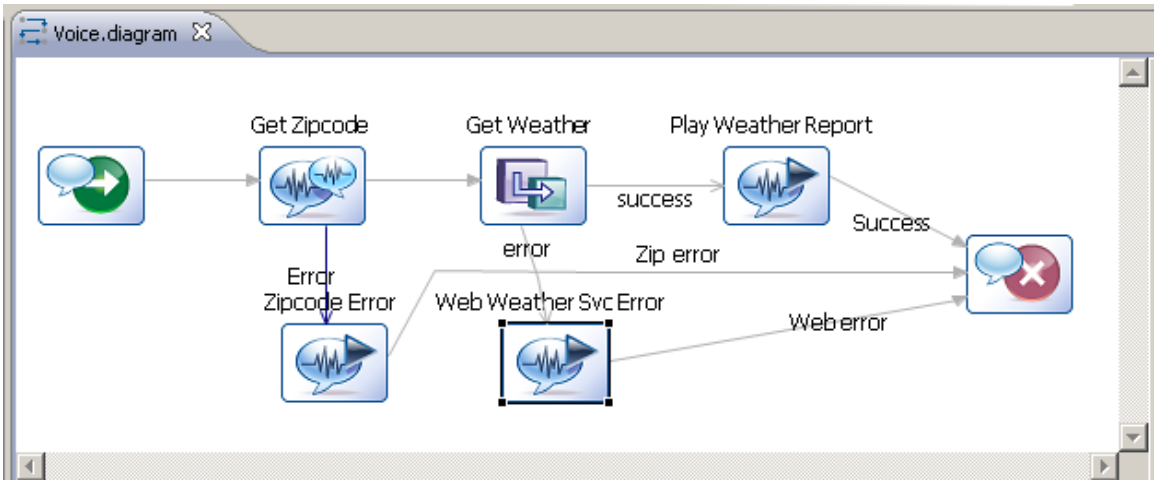


Figure 5.9 - Final Web Weather Diagram

This completes the web services weather recipe.

13. Package your project.
14. Upload the WAR to a runtime system to test that it will play the weather report.

Recipe 6: DTMF Error Recovery

Contributed by Melissa Tatum.

Approximate time: 60 minutes, depending on prior experience

What you will learn with this recipe:

- What Errors Occur in a DTMF Menu.
- What Error Recovery Does
- How to Configure the DTMF Menu Component to Handle Error Recovery
- Global Event Handling
- Defining Maximum Error Counts for the DTMF Menu component.
- Event diagrams – Purpose and Usage.

What the DTMF Error Recovery Recipe Does

The Menu is one of the most important parts of a dialog system, as that is where we determine what a caller wants to do. Recipe 2's NewsWeatherSports (NWS) showed how to present a caller with a menu of choices, using IC's DTMF Menu component. The caller makes a menu selection using the touchtone keypad on the phone.

That first NWS recipe demonstrated the basics of menu selection, but that recipe wasn't very user-friendly. If the caller made a mistake entering their selection, or waited too long to enter their choice, they are summarily tossed out of the application with an abrupt "goodbye". There are no second chances with that recipe. We must do much better, if we want to build an application that people will want to use.

This recipe will show you how to modify the original NewsWeatherSports recipe to make it more user-friendly. We want to give fumble-fingered and slow-responding users extra chances to get their keyed entries right, with some helpful comments along the way. We will try to show "best practices" in dealing with errors that callers often make when using touchtone menus. We will focus on how to implement those best practices when using the DTMF Menu component. To understand the error-handling aspects of the DTMF Menu component, we will need to understand more about event processing in components, specifically the Error, Menu Error, and Menu Timeout events.

Also, this recipe will explain event handlers and event diagrams, which is how IC deals with common conditions that may occur multiple times in a single application. Instead of handling the same condition over and over in your diagrams which causes lots of diagram clutter, IC provides a mechanism to route commonly-occurring conditions to a central event handler, where the response to a specific common condition can be defined once, in one place. Our goal with be to create a user-friendly DTMF menu dialog structure that you can reuse in any application.

What does Error Recovery Actually Mean?

When presented with a touchtone menu dialog, callers make two common errors:

- Caller makes a selection that is not in the list of valid choices. For example, suppose the caller is presented with three choices: "Press one for News, two for Weather, and three for Sports". If the caller selects seven, this is an invalid selection, or a NoMatch error. A NoMatch error will occur any time that a caller makes a selection that is not in the list of valid choices as defined in the DTMF grammar.
- Caller fails to make any selection at all. Typically, there is a maximum entry timeout associated with any DTMF menu. If the user has made no selection, and that timer expires, we have an error. That scenario is an entry timeout, or in the VXML vernacular, a NoEntry error.

When we design voice response applications, we want to make the caller's experience as pleasant as possible. Therefore, if a caller makes an error, give the caller helpful information about what went wrong, and then let

them try it again. If we can present pertinent, helpful messages to the user when they make an error, and then propose a retry, the user may be able to recover from his error situation without too much frustration. The unhelpful first-error "goodbye" that we designed in our first Menu recipe, is a good way to create an unsuccessful, and very unhappy caller.

Unfortunately, some callers just can't seem to get it right, even with the best of help. To avoid tying up a confused caller in a never-ending loop of help messages and retrials, we must at some point give up trying to help. A good general rule is to give a caller three times to correct an error. If we get to that point in an error recovery dialog, it may be best to enlist a live agent on the call to see if they can help, or go to some other plan to discover the caller's requirements. In this recipe we will show you how to define a unique exit from the DTMF menu component if a maximum error count is reached. You can make that exit path do whatever makes sense, in your particular context.

In our NWS recipe, the caller hears the following spoken menu:

> "For News press 1. For Weather press 2. For Sports press 3"

If the caller makes an error and presses the 'seven' key on their phone, gently inform the caller that seven is not a valid option. Explain that the valid options are one, two, or three. In the other caller error scenario, the caller fails to make any selection at all after waiting an appropriate time. In this case, firmly inform the caller that they should press the touchtone key representing their selection, now. The Menu component has all of the logic to play these help prompts, built-in. All you have to do is configure the prompts.

If the caller still has a problem after a second try at it, give them a final, third chance to get it right. If they mess that up too, it's likely that they need more help than our automated agent can provide. In this situation normally hand them off to a live operator, to see what their problem is. If you don't happen to have a live agent handy, for this recipe you can just substitute a prompt saying "this would transfer to a live operator" when that path is taken.

The DTMF Menu component has built-in error features that can detect the errors described above which can occur during a DTMF menu dialog. It can play unique error messages for the different types of errors that a caller might cause. It also keeps track of the total number of each type of error that occurs, and it will perform specific actions if a preset maximum of errors is reached. The menu component will handle much of the lower-level work of error handling, so you the developer can focus on the main parts of the dialog design. The DTMF Menu component is one of the more powerful components in the IC toolbox.

So how do we add these helpful error dialogues to our NWS recipe? That is the main subject of this recipe.

Developing the Call Flow

To begin, create a new project for this error-handling version of the NewsWeatherSports recipe. However, we won't need to build this recipe from scratch. Our original DTMF Menu recipe (#2) has the basic menu stuff in it that we need to start building the error recovery process. Just copy the original NWS DTMF Menu recipe to this new project, as a basis of our new error recovery recipe. This will save us from having to rebuild the original DTMF Menu recipe and re-enter all of the properties in the original recipe.

Create a new ICS Application Project in IC, and name this new project **ERNewsWeatherSports**, for the error recovery version of our DTMF menu recipe. Then create an ICS Voice diagram. For specific information about creating a project and voice diagram, refer to Recipe 1's **"Hello World"** recipe. Then, we'll use the original NewsWeatherSports application as our starting point, and then just add the error recovery dialog to it.

To prepare this new Voice diagram for the NWS components we are going to insert, stretch the Voice Begin and Voice End components apart, and delete the connector. You will be pasting the old NWS recipe components in between the Voice Begin and End components.
Now go back and open the original NewsWeatherSports recipe. Then launch that recipe's Voice_NWS.diagram from the Package Explorer window. You should have the old recipe's Voice diagram showing in the main diagram window. Select the NewsWeatherSportsMenu component, as well as the four Play Components (News, Weather, Sports, and Goodbye) all at once. You can select all of the components by starting at one

corner and dragging a box around the components you want to select, or use Control-click to select/unselect each component individually. You don't need to select the green Voice Begin or red Voice End components, as those components are already in our new project.

After the components are selected (you will see that they have a black box around them), right click the mouse on one of the selected components and select **Edit > Copy** (see Figure 6.1). Then, go back to your newly created Error Recovery NewsWeatherSports application and paste the components into the Voice.diagram (see Figure 6.2). NOTE: Notice the red and green arrows on the voice.diagram. This shows the direction of the components. In this case, NewsWeatherSportsMenu is highlighted. Direction into this component is via Voice Begin. Direction out of the component is via Play components – News, Weather, Sports, or Goodbye.

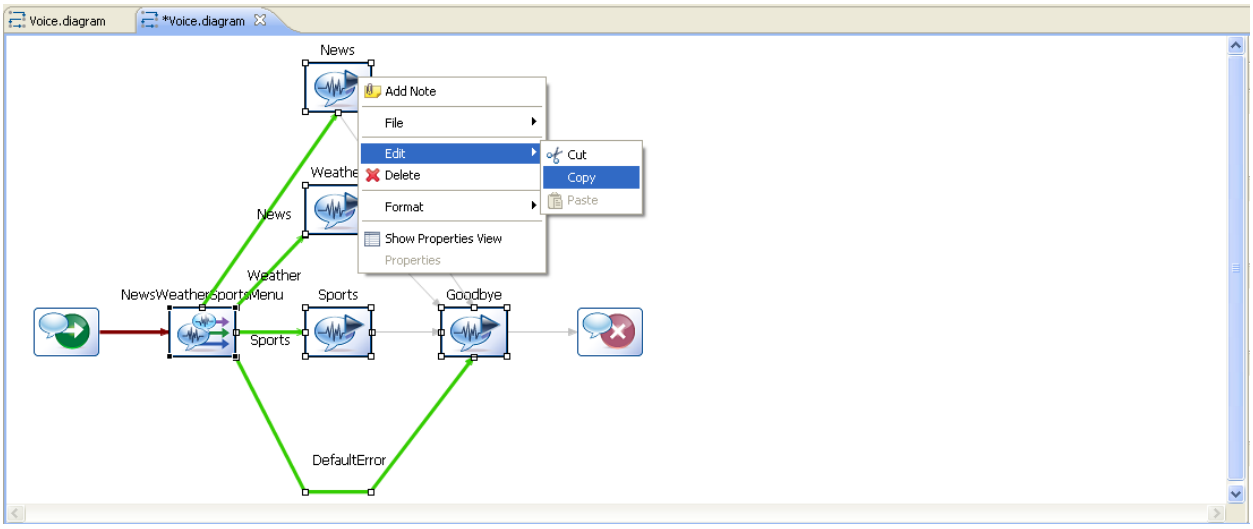


Figure 6.1 - Copying Selected Components from NewsWeatherSports Voice.diagram

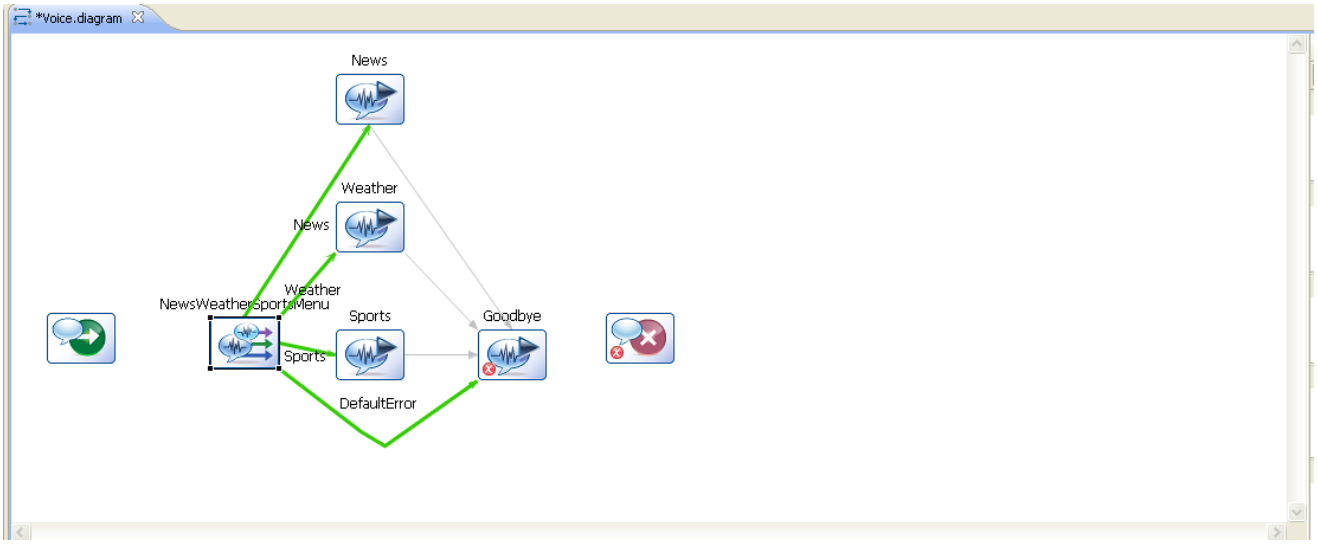


Figure 6.2 - Components Pasted into ERNewsWeatherSports Voice Diagram

This new Voice diagram already has all of the main menu prompts and properties defined, since they were all copied from our original DTMF Menu diagram. All we have left to do is add the error recovery dialog flow, and we will be done. Most of this work will be just setting the properties in the NewsWeatherSports Menu component. Notice again the green arrows on NewsWeatherSportsMenu. Notice also that there is no red arrow. The red arrow will appear after VoiceBegin and NewsWeatherSportsMenu are connected.

Defining Error Event Messages

The first step in designing an error recovery dialog is to define the individual messages that will be played when a caller makes an entry error. The messages we define will be played whenever the caller makes a NoMatch or NoInput error. The DTMF Menu component allows us to play different error messages to the caller for each error type. Also, different messages can be played on subsequent occurrence of the same error, by the same caller. In all cases, a unique error message can be played on each error occurrence with a caller, until the maximum error count is reached.

As you will see, the DTMF Menu component has a lot of flexibility as to how you can build an error recovery dialog. We need to keep that in mind as we design the error help messages. It is a good idea to think through just what you want to say, and when you want to say it, before starting to configure the DTMF Menu component properties.

Best practices recommend that we limit caller error retries to three, in most menu situations. If the caller makes a mistake, we want to make that first help message brief. If the caller makes the same error a second time, that help message can be a bit more detailed. If the caller makes a third mistake, the third and final error message should explain basically that we are giving up, and at this point the caller will exit the DTMF menu component, typically to be handed off to a live agent.

Later in this recipe, we will show you how to set the maximum number of entry errors to 3, which is what best practices recommend. For now, we will build all of our error recovery prompts assuming that the caller will get up to three tries to get their entry right. If the caller makes a valid entry at any time during the error recovery process, take the caller out of the menu component, and on to the appropriate menu path they have selected. If the caller makes three errors without a valid entry, take one of several Max Error exits from the Menu component, depending on what kind of errors have been made.

In this best practices example, we want to present a different help prompt each time the caller makes an error in that single menu. The first error by the caller will trigger a prompt that provides a basic description of the problem, and then brings the caller back to the selection state in the dialog. If the caller makes the same entry mistake again, play a different prompt that gives a bit more detail about the error that the caller made, before letting them select again. This will hopefully help the caller to get the correct entry in the second try.

If the caller still fails to make a valid entry on the third try, exit the DTMF Menu component, and go for more expert help. This technique makes the error recovery dialog somewhat complex, but this is the most user-friendly process. This technique is called “escalating prompts”, and this escalating capability is built into the DTMF Menu component.

To build an escalating error recovery dialog, you simply have to define a prompt group in the DTMF Menu component for each of the prompts you want to use. In our case we will have two NoMatch prompts, and two NoEntry prompts for the basic escalation steps. In each case, the first prompt for an error will be brief, and the second prompt for the same error will have more detail. If the caller makes a third error during the menu process, the maximum error count event will occur. Once this third error event occurs, we will use an event handler process that is external to the DTMF Menu component to deal with that condition. The prompts that are played after the third error are defined in the event handler outside of the DTMF Menu component.

Here is an example of the escalated prompts for when a menu entry error occurs:

1) First NoEntry Error – “Sorry, but I did not receive any input.”
2) Second NoEntry Error – “Sorry, but I still did not receive any input. Please make your selection now.”
3) First NoMatch Error – “Sorry, but that is an invalid selection. Please try again.”
4) Second NoMatch Error – “Sorry, that is an invalid selection. Your options are 1 for News, 2 for Weather, and 3 for Sports.”

We need to create four Prompt Groups to hold these four error prompts. Then we can configure the conditions under which each prompt will be played. All of this can be done within the DTMF Menu Properties view.

To create the four Prompt Groups, go to the Properties view > General option > Prompts tab > **Event** subtab. Then right-click in the Event area, and click **Add a new event prompt group**. (See Figure 6.3).

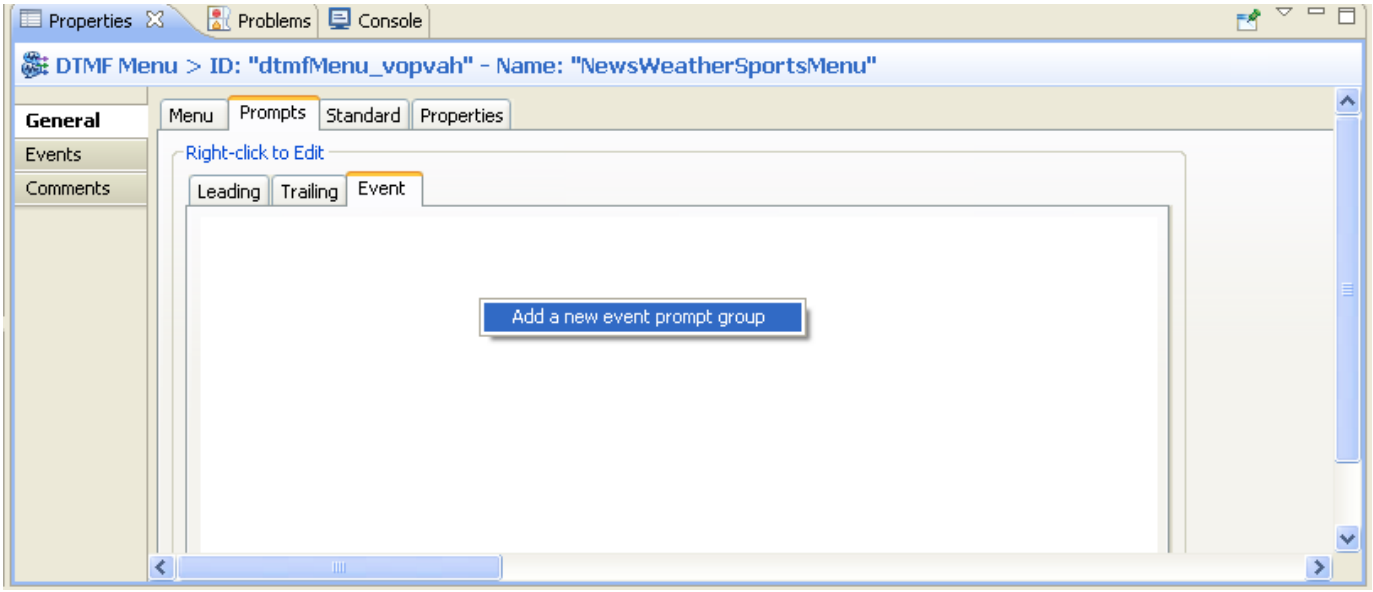


Figure 6.3 - Adding an Event Prompt Group

The Prompt Group name will automatically be generated for you.

When a Prompt Group is initially created in the event window, you won’t see that groups’ attributes to the right of the prompts box. You must click on a Prompt Group name in the box to highlight it, and then the attributes for that group display. Click on our first Prompt Group name, **Prompt Group – NoInput_1** and that group’s attributes display. Note that the Prompt Group is automatically named **NoInput_1**. The name is defined by a combination of the Scope (error type) attribute, followed by the Prompt count. Changing the Scope and Prompt Count attributes in the fields on the right will change the Prompt Group name in the Event box.

You should set the attributes for the first Prompt Group as shown in figure 6.4.

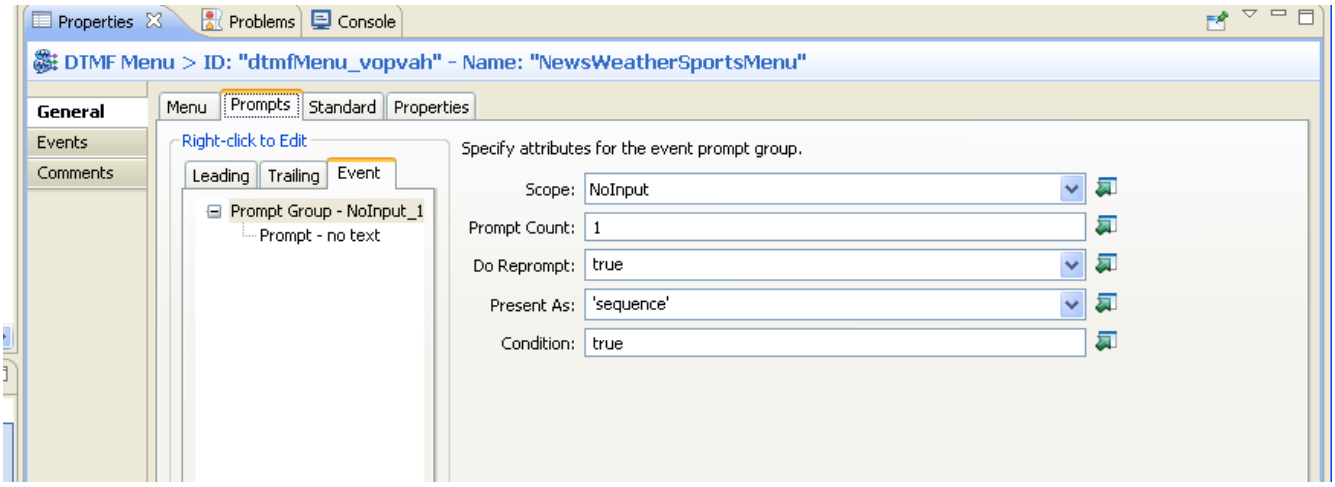


Figure 6.4 - NoInput_1 Prompt Group

The Scope attribute for this first prompt is set to **NoInput**, which specifies the type of error that will launch this prompt. The Prompt Count is set to **1**, specifying that this particular prompt group should be played only on the first NoInput error that occurs. The Condition attribute of **true** makes this error prompt play every first time that the NoInput error occurs. A more complex Boolean expression for this attribute can vary this rule.

Setting the Do Reprompt attribute to **true** means that the system will play the NewsWeatherSportsMenu prompts again for the caller after the error prompt. This includes the leading prompts as well as trailing prompts. If you don't want to play the leading prompt on each error each time, place the leading prompts in a Play component in front of the Menu component, and take out the leading prompt message from the DTMF Menu component. In this example, we are choosing to play the leading menu prompt each time, since it is a part of the DTMF Menu component.

After you have finished setting the attributes for our first prompt group, you should define a specific prompt for that group. You will notice below Prompt Group-NoInput_1 , there is Prompt - no text defined (see Figure 6.5).This will create the location for our first error prompt, which will be played on the first NoInput error by the caller.

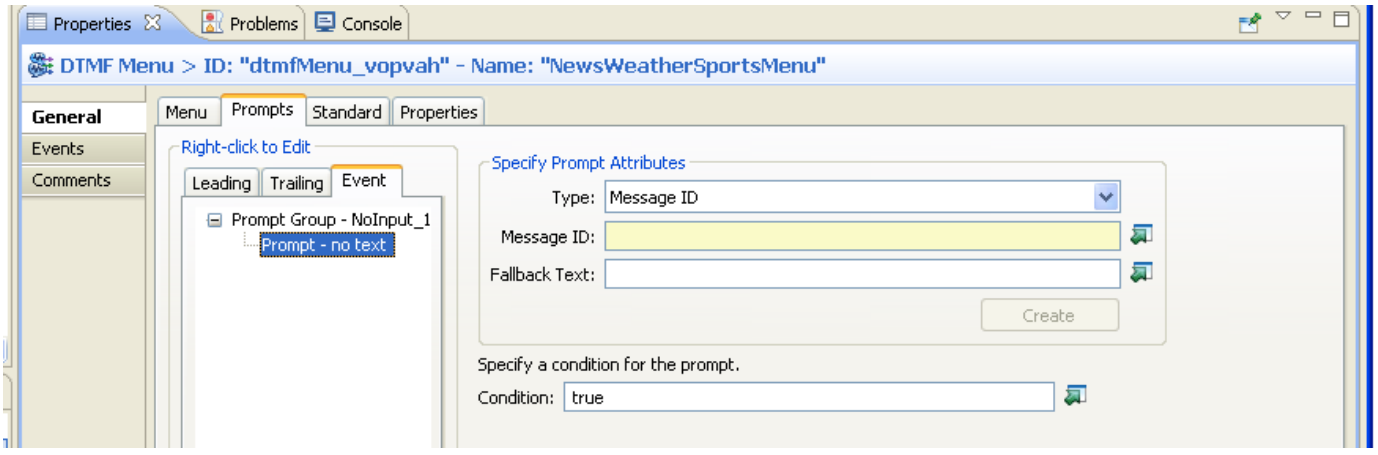


Figure 6.5 - Defining a new Event Prompt

Figure 6.5 shows the newly-created prompt in the first Prompt Group. We still need to define the specific properties of that prompt, particularly what it will play. Just like defining prompts on previous recipes, highlight the **Prompt** under Prompt Group – NoInput_1 and define the message to be played in the Prompt Attributes fields. Notice that we are again using TTS as in previous recipes to avoid having to record actual audio prompts. Therefore for Type select **TTS** from the dropdown box rather than using the default Message ID. For this first NoInput error condition, use single quotes and set the prompt expression to: 'Sorry I did not receive any input'. See Figure 6.6.

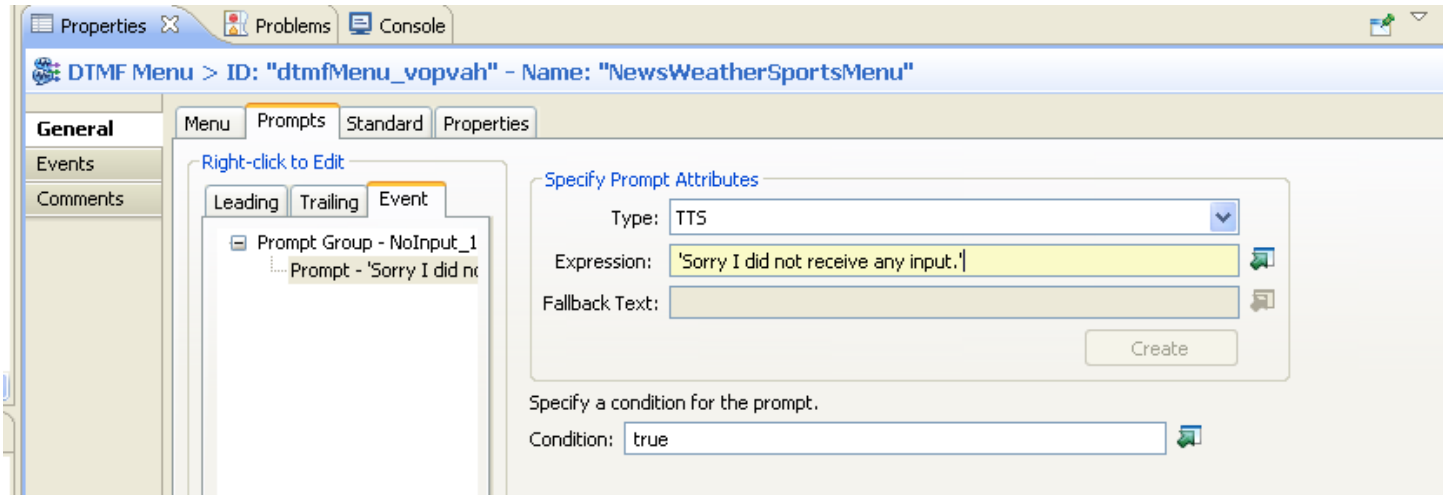


Figure 6.6 - Completing the New Event Prompt

Now go ahead and create the second **Prompt Group** using the same process as in Figure 6.3. This prompt group will also have a NoInput event Scope, but in this case, the Prompt Count attribute will be **2**. This second prompt will only be played if the caller makes a second NoInput error on the same menu. You can refer back to our previous discussions to see what that actual text for each prompt should be.

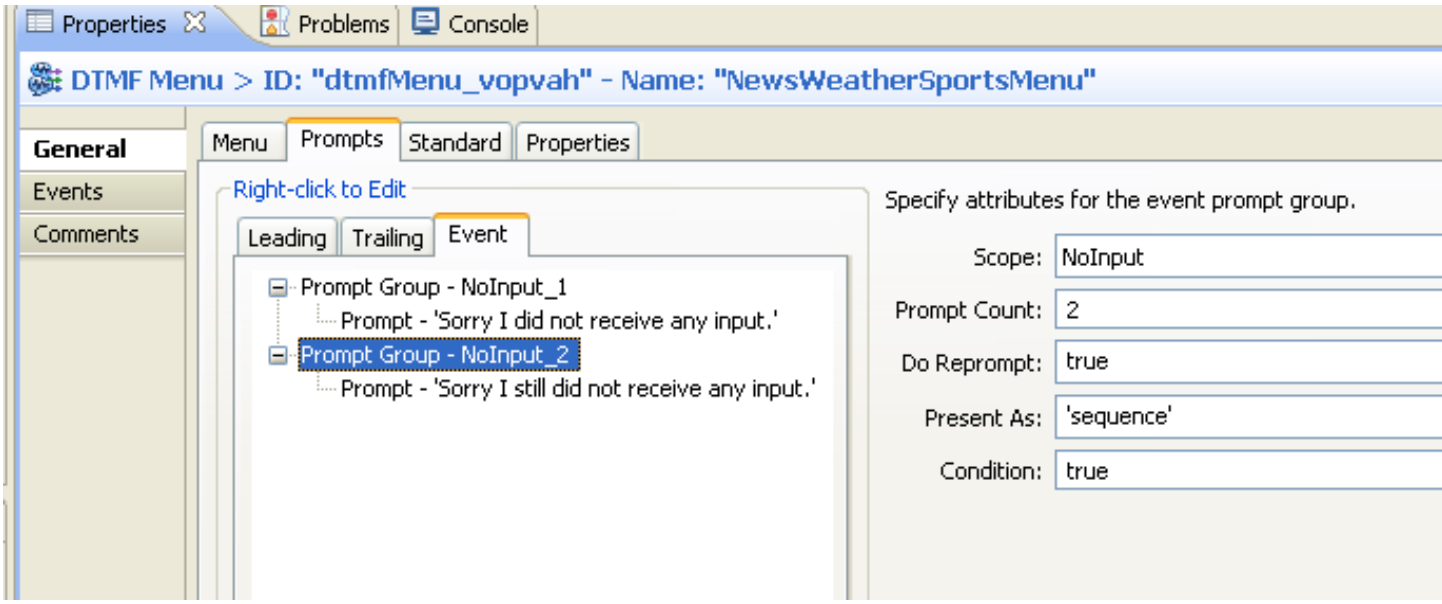


Figure 6.7 - Completing the Second Prompt Group

You can finish up the prompt groups for the NoMatch errors as well. The completed set of error prompt groups will be similar to Figure 6.8. You might ask why we're not defining three prompt groups, since we have specified that we will set the maximum number of errors to be 3 for NoInput and NoMatch. That is because we are going to specify that a third consecutive input error will take us out of the DTMF component altogether. We will cover this step in the next sections.

Figure 6.8 - The Completed Set of Error Prompts

For now, we can see that if the caller keys in an invalid option, Prompt Group – NoMatch_1 will be played. The second time the caller inputs an invalid option, Prompt Group - NoMatch_2 will be played. The third time, the caller "keys in" an invalid option, we should reach the Maximum Error count and exit the DTMF Menu component. We don't need to define a third error prompt in the Menu component, as we will play that prompt in an external event handler, which we will describe in the next section.

The Event Handler

Now that we have defined the initial error processes and messages for our menu, we need to define what happens when the caller can't get their menu selection right, even after three tries. We might want to transfer the caller to a live agent, or perhaps verify that this customer has enough value to merit a live agent. Perhaps we could send the caller to a separate speech input menu and prompt the caller to speak their choice, instead of using the phone keypad for the selection.

No matter what we do to recover from multiple-menu errors, it is likely that we will want to define a similar process for most of the menus in our application. In this recipe we only have one DTMF Menu, but in a large application there can be dozens of menus. It would be nice if we could define a standard maximum-error-handling process for all of the menus in an application, in one place. Otherwise, we would have to define this process for each menu, which could clutter up a voice diagram significantly. Better yet, it would be great if IC would provide one central place to put handlers for all kinds of commonly occurring events, not just errors. That would keep our diagrams tidy, and make maintenance much easier.

IC has a solution for these issues, and it is called the event handler. Errors aren't the only reason to call a common handler routine, which is why we call it an event handler rather than an error handler. In our DTMF Menu recipe, all of the events we want to catch are essentially multiple-error retry conditions. However, that is only a few of the common events that might occur in an application, and we would like to centralize all of these types of events in a common place. That place is the Event Handler.

Configuring the DTMF Menu Max Error Counters

Click on the DTMF Menu component's **Properties** view > **Events** option (on the left). You should see four error counters - Error, MaxNoMatch, MaxNoInput, and MaxErrors. Remember that these four specific parameters are counters, not direct routine calls. They won't do anything but count the occurrences of a specific error type, until a maximum count is reached. However, when that count is reached, they will trigger a call to the event handler that we select.

This section of our recipe will define how these counters work, and how to define the appropriate event handler that is executed when a maximum count is reached. In subsequent sections we will describe how to set the maximum counts for each counter, as well as how to configure the specific event handler that is called when that maximum count is reached.

Here are the definitions of each counter:

MaxNoMatch- This counter increments each time the caller enters an option that is not a valid menu option. If the caller enters 7 in the NWS recipe for example, that will increment the MaxNoMatch error count.

MaxNoInput- This counter increments each time the caller fails to enter a selection before the entry timer times out. If a caller doesn't respond to the menu prompt with any input, that event will increment the MaxNoInput counter.

MaxErrors- This counter counts the sum of MaxNoMatch and MaxNoInput errors. Either type of error will cause this counter to increment.

Error- This counter counts errors that are neither a MaxNoMatch nor MaxNoInput, error. These are typically system or programming errors. For example, a TTS error, or a missing audio prompt file, will count as this type of error. These kinds of errors are usually catastrophic, since it means that the system can't play a specific prompt. You won't get to set a maximum error count for this condition. The max count for this condition is forced to one, and you can't change it. You will be able to select an event handler for this condition, however.

Figure 6.9 shows the DTMF Menu **Properties** view > **Events** option.

- The Error event covers errors that are not caller-input errors. Hopefully, you won't encounter these kinds of errors very often.
- It is important to understand that the MaxNoMatch, MaxNoInput, and MaxErrors events will only occur when a pre-set maximum error count has been reached.

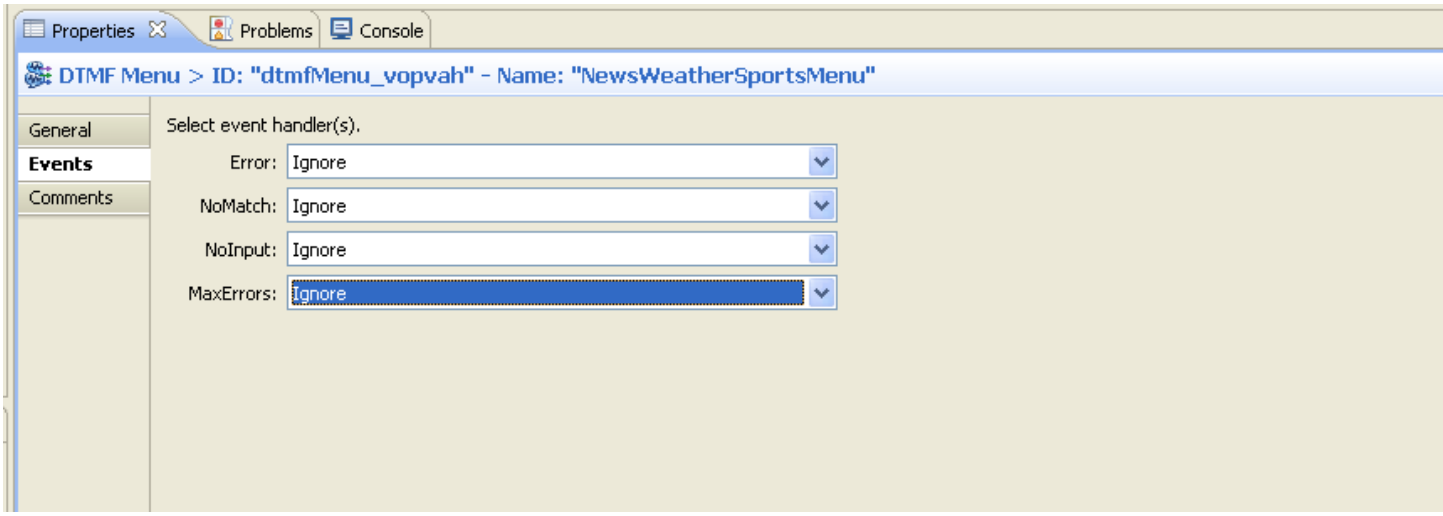


Figure 6.9 - News Weather Sports Menu Error Events

By default, all of these error counters are set to **Ignore**.

Now that we want to be more user-friendly, we can use the dropdown event handler selections to uniquely specify how to process each error type. For each specific event counter, you should define the particular event handler that you want to execute when that maximum error count is reached

All of the event counters have the same set of options:

- **Ignore**—When a maximum count is reached, the DTMF Menu component exits on the default exit path. You would only use this if you didn't want to do anything special on the max error condition, which isn't normally good practice.
  Note: Since the maximum count in our first menu recipe was set to one, we simply took the default exit path out of the menu component on any error. This wasn't very user friendly, since the default exit simply played "Goodbye" and hung up. However, it kept things simple for our first menu recipe.
- **Inline**—Creates a new exit path from the DTMF Menu which allows us to build our own error handler, right on the voice diagram. We normally want to avoid this scheme also, since it would clutter up our diagrams with all of these individual error handlers. We will normally always want to handle the max error condition for menus in the same way, with every menu. We would use the inline option only if we have a unique event-handling routine we want to execute on one specific menu.
- Error
- Menu Error
- Menu Timeout
- Collection Error
- Collection Timeout
- Host

Except for Ignore and Inline, all of these selections are the name of a predefined Event Handler routine that we can use.

In reality, there is only one event handler with multiple entry points, which allows you to see all of the global event processes in one diagram. This makes it easy to change the behavior of a particular event globally, without having to search the diagrams for all of the occurrences of that event process. These standard events have been selected from extensive experience to be events that commonly occur in most every application.

Specifying the Event Handlers for Each Error Condition

At this point, we need to define the specific event handlers that will be executed when any error counter reached its maximum in the DTMF Menu component. Remember that the DTMF Menu component will keep the caller in its internal dialog flow until they make enough errors to reach one of these max error counts (or the caller makes a correct entry). So if we reach the point in the dialog where an error counter has hit the maximum, the caller has already caused several (or in our case, 3) errors.

For each of these error counters, use the dropdown list to select an event handler that will be processed if that counter exceeds its maximum count. The IC event handler diagram will perform a specific task for each predefined event, and then exit.

For this sample recipe, use the Menu Error, Menu Timeout, and Error event handlers. Since these three events can happen in any menu in our application, the best practice is to have a common handler for all of them:

For Error select **Error**
For MaxNoMatch select **Menu Error**
For MaxNoInput select **Menu Timeout**
For MaxError select **Menu Error**

This process gives us a unique error handler for two error counters (Error and Timeout) and a third exit for either NoMatch or MaxError counters. Figure 6.10 shows the event handlers selected for best practices.

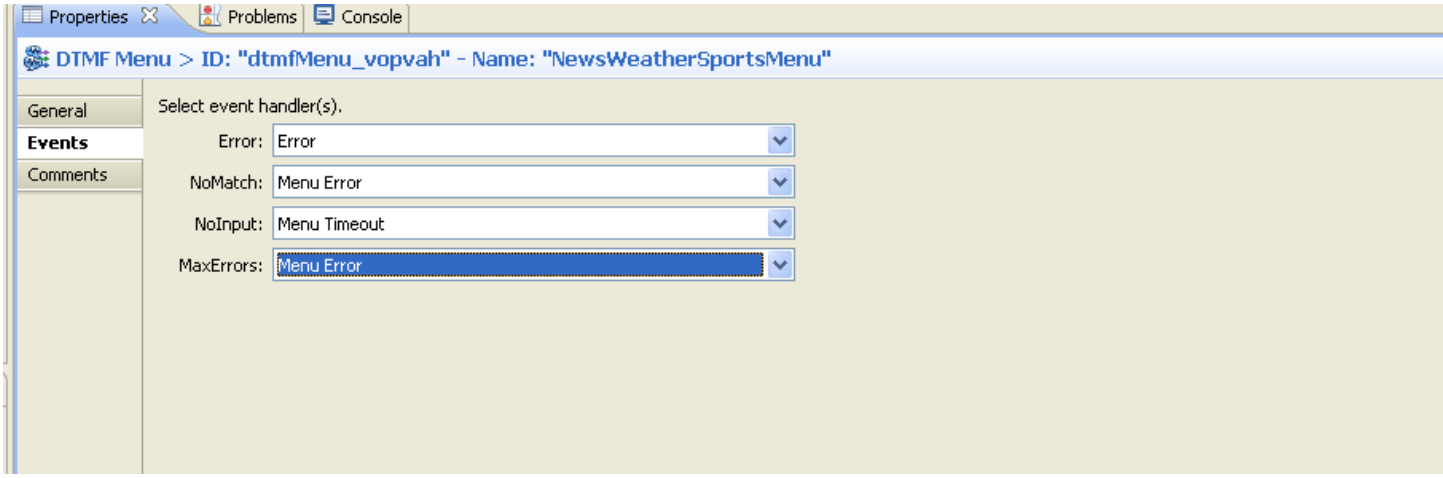


Figure 6.10 - Selecting the Event Handlers for the Errors

This configuration allows us to have unique error-handling processes for similar classes of errors. If we reach this point in the dialog, a NoInput or timeout error means that the caller has never entered anything during the complete menu dialog, even after multiple help prompts. A Menu Error means that the caller has entered something, but it was wrong – three times. There may also have been some timeouts in this process, but at least the caller seems to be trying. A pure Error means that there was an error with the prompt mechanism, or some other system error, that was not caused by the user. Best practices dictate that these three conditions should cover the primary error states in any menu, which is why IC has predefined them.

Setting the Maximum Error Retry Limit

Before we can start defining what our event/error handlers will do, we need to set one more thing in the DTMF Menu component – the error retry limits. To prevent a key-entry-challenged user from endlessly attempting to make an erroneous selection, we need to limit the number of user-input errors that can occur. We typically give the caller three times to get their selection right. If the caller fails to make a valid selection after three tries, they most likely need help from a live person, so we would typically transfer the caller to an operator.

To set the retry count limits, select the **NewsWeatherSports** DTMF Menu component in the voice diagram. In the **Properties** view > General option, select the **Standard** tab. See Figure 6.11. This is where you can set the properties for the Max Errors, Max NoMatch and Max NoInput counters.

In our original NWS recipe these maximum counts were all set to **1**. Change the three max count values to **3**, to follow our best practices for DTMF Menu error handling. The caller will stay in the DTMF Menu component until he either makes a correct entry, or reaches one of the max error limits.

Setting the max error counts as shown will allow the caller to have three NoMatch errors (where an invalid option is chosen), or three NoInput errors (where the caller experiences a timeout – basically caller doesn't "key in" anything), or a combination of these errors that add up to three, which we specify in MaxErrors. If one of these maximums is reached, the call flow leaves the DTMF Menu component through the error exit and goes to the event handler.

To get a better understanding of MaxErrors, here's an example. If a caller has two NoMatch errors and one NoInput error, then Max Errors will now be three. At this point, we will execute the processing that is defined for MaxErrors, which we will define in the event handler sections. An easy way to understand MaxErrors is to realize that this is just a combination of NoMatch and NoInput errors encountered. Figure 6.11 shows the completed error count entries for best practices error recovery.

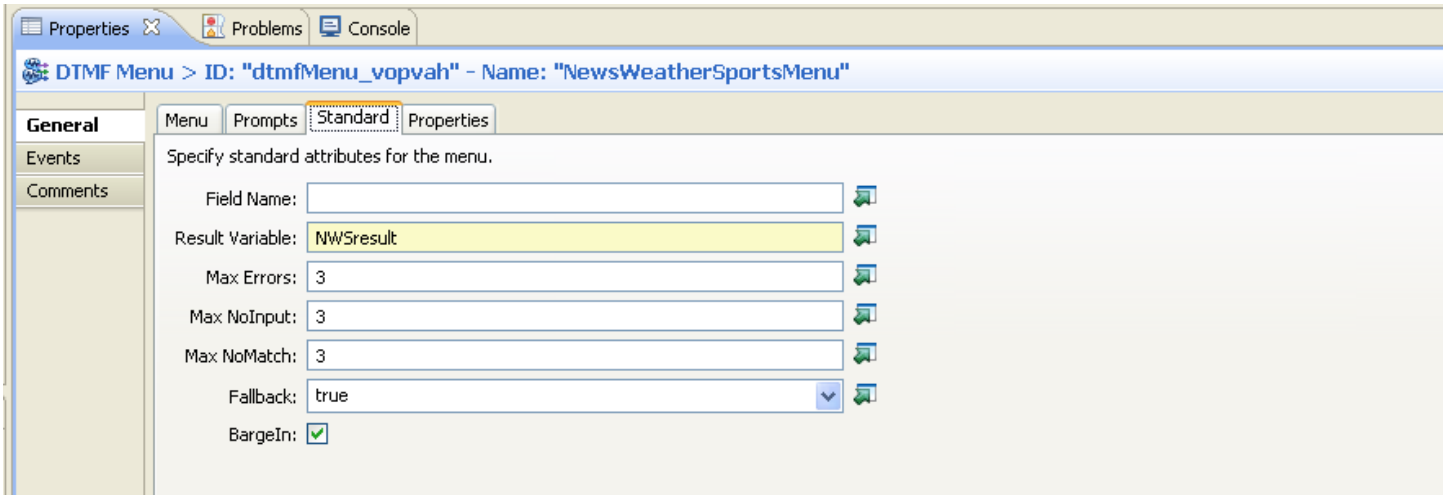


Figure 6.11 - Defining the number of Errors

Completing the Event.Diagram

Just because the IC event handler diagram has predefined entry points for standard dialog events doesn't mean that what the event handler does is also standardized. In most cases any of these max-count menu errors would cause a simple explanation prompt to be played, followed by a transfer to a live operator, but it is easy to add more comprehensive dialogs in the event handler.

The Event.diagram is located in the voice folder along with the Voice.diagram. Double-click on the diagram file in the Package Explorer to display the Event diagram in the main window. See Figure 6.12. The default Event.diagram only has three components – an Event Process component, an Event Entry component, and an Event Return component.

Click on the Event Process component (Gear) to select it. Look at the Properties view to see the Event Process components' properties. See Figure 6.12. The Event Process has six Event Names properties by default: Error, Menu Error, Menu Timeout, Collection Error, Collection Timeout, and Host. These names comprise the six standard entry points for the Event diagram. The default actions for all of the Event Names are defined as ErrorAction by default. For this recipe, we will only be concerned with three Event Names: **Error**, **Menu Error**, and **Menu Timeout**. Remember that these are the names of the events that we defined in the NewsWeatherSportsMenu.

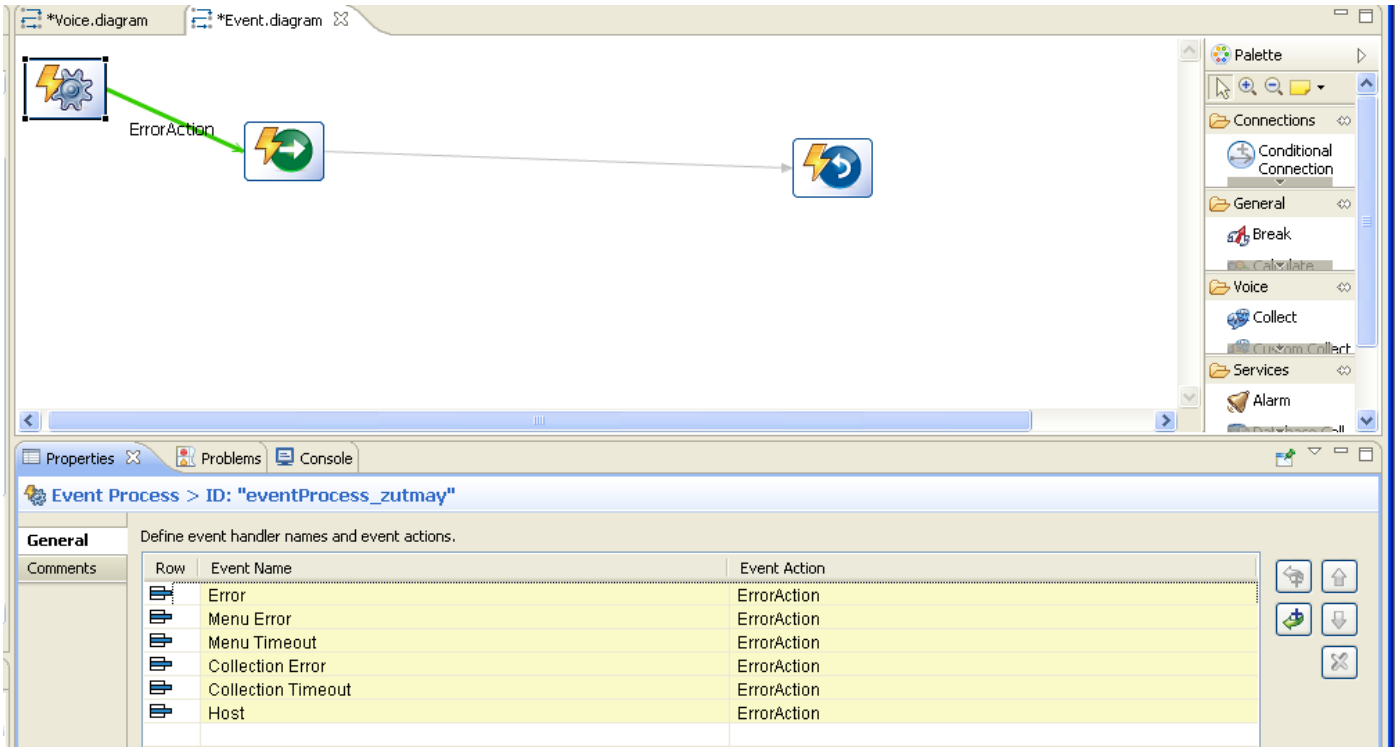


Figure 6.12 - Default Event Process in the Event Diagram

Note: For more complex applications you don't have to stick to the six standard entry points. These standard Event Names are not the only event handlers we can define. The Event Process component allows you to add more custom event names and EventActions if you like.

Begin building our custom event handlers by adding two additional **Event Entry** components onto the Event.diagram, to the right side of the diagram. The Event Entry component is located on the **Palette > Events** drawer. This gives us three Event Handler entry points total, which will cover the three different error handlers we want to create. See Figure 6.13.

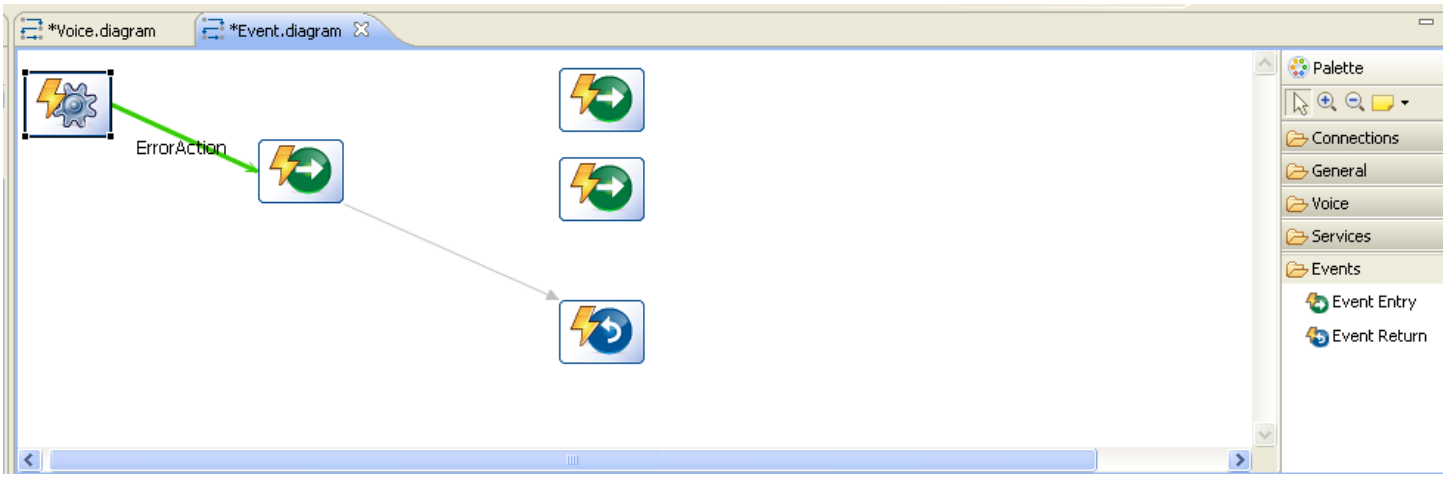


Figure 6.13 - Adding Event Entry Components to Event Diagram

Defining the New Event Entry Components

Now that we have added in the two new Event Entry components, integrate them into the Event diagram. Select the **Event Process** component which is the first component in our Event diagram. The Event Process component properties allow us to configure the Event Action associated with each Event Name. Figure 6.12 shows the Event Process component properties where we can associate the entry points in the Event Process component to specific Event Entry components.

Remember for this recipe, we are only concerned with three events: Error, Menu Error and Menu Timeout. For two of these error events we will define a different Error Action, rather than using the default of ErrorAction. The default ErrorAction will execute the generic error handler deals with the multitude of system and other (hopefully rare) errors that may occur, for which we don't have a specific strategy.

For Menu Error, type **MenuNoMatch** to replace Error Action. For Menu Timeout, type **MenuNoInput** to replace Error Action. There are no predefined names for these actions, so you must type the name. See Figure 6.14.

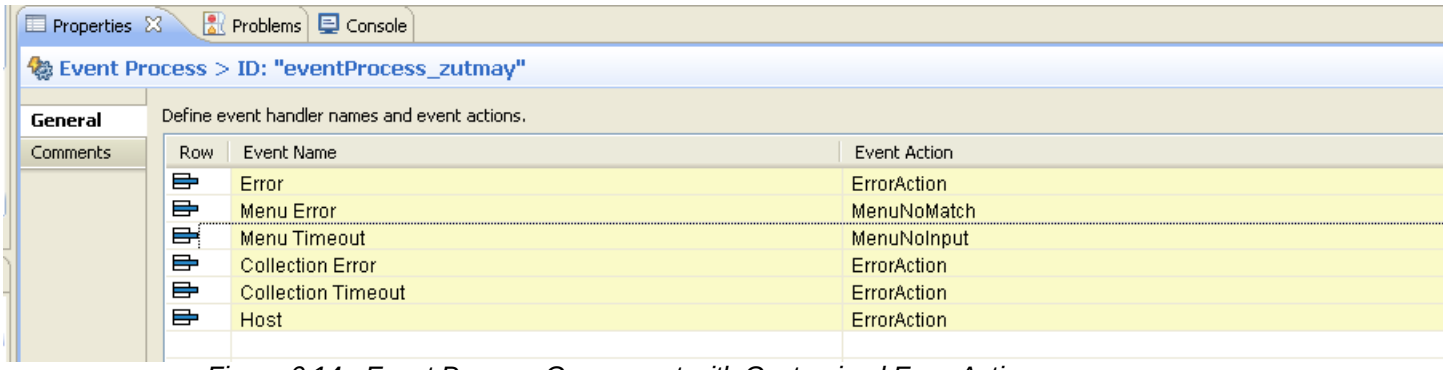


Figure 6.14 - Event Process Component with Customized Error Actions

Once the Event Actions for MenuNoMatch and MenuNoInput have been defined, they can be used in the event diagram. Each unique Event Action will be presented to us as a possible connector from our Event Process component when we begin connecting it to other components in the event diagram.

Click on the **Event Process** component and connect it to one of the newly-added **Event Entry** components. You will see a connection selection menu when you add the connection (See Fig 6.15). Select the **MenuNoMatch** option. Then you can connect the **Event Process** component to the other **Event Entry** component and select the **MenuNoInput** selection.

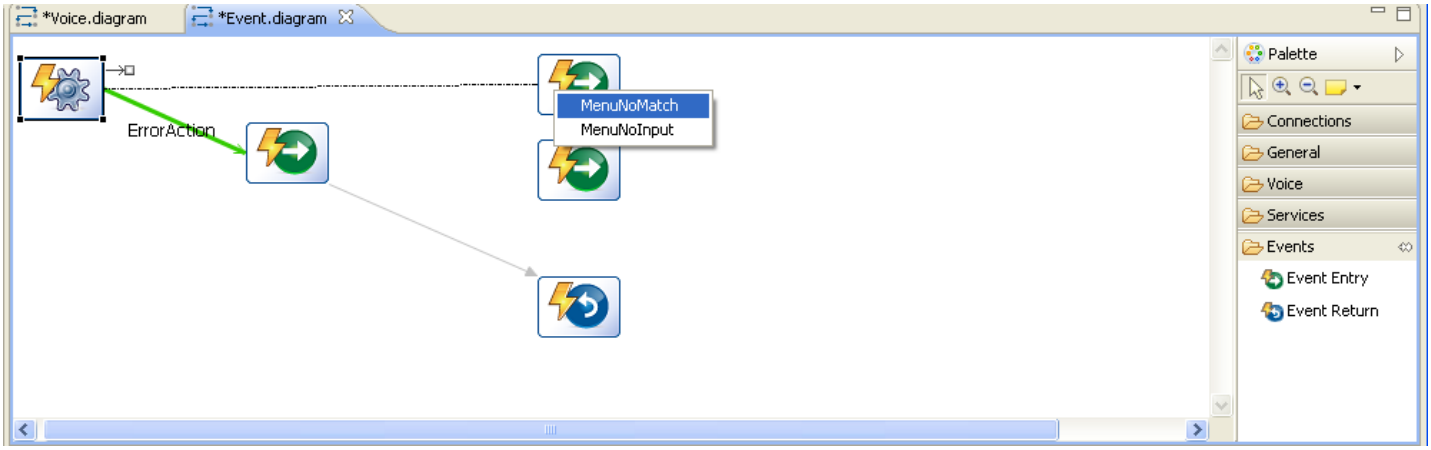


Figure 6.15 - Connecting Event Entry components to Event Process

Label the Event Entry components at this time with the appropriate labels: **NoMatchError** for the NoMatch Event Entry component, **NoInputError** for the NoInput Entry component, and **SystemError** for the system error entry component. See Figure 6.16.

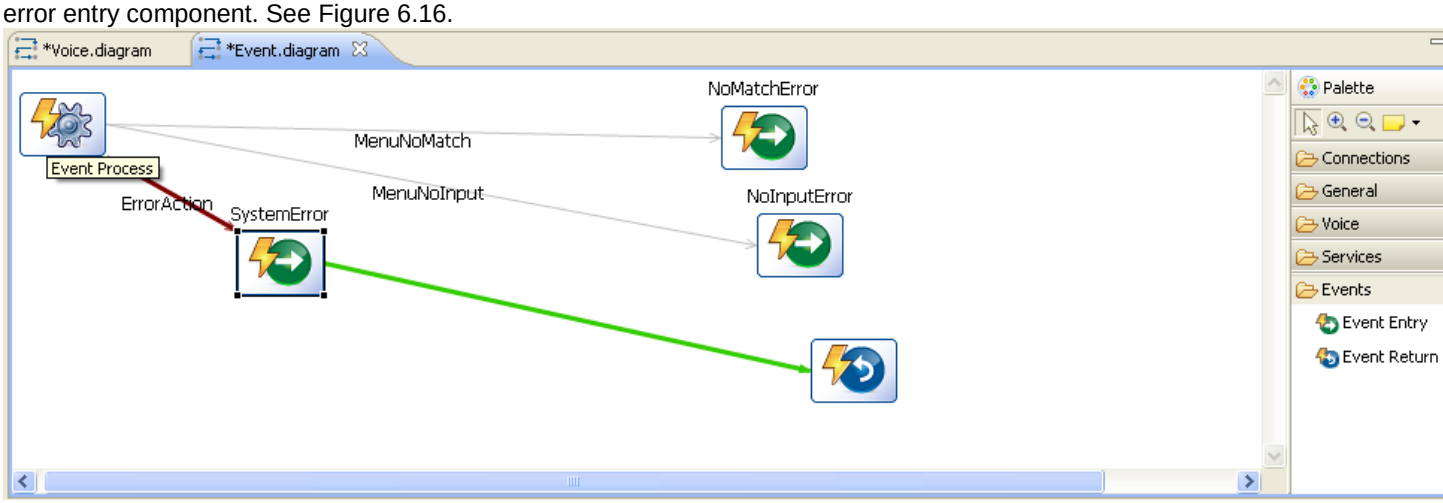


Figure 6.16 - Connections Completed

It is good practice to play a message to the caller about what happened after a MenuNoMatch or MenuNoInput event is encountered on the Event diagram. In that way, the caller will know what the problem is, and what to expect. So add **Play** components after each of these events (See Fig 6.17).

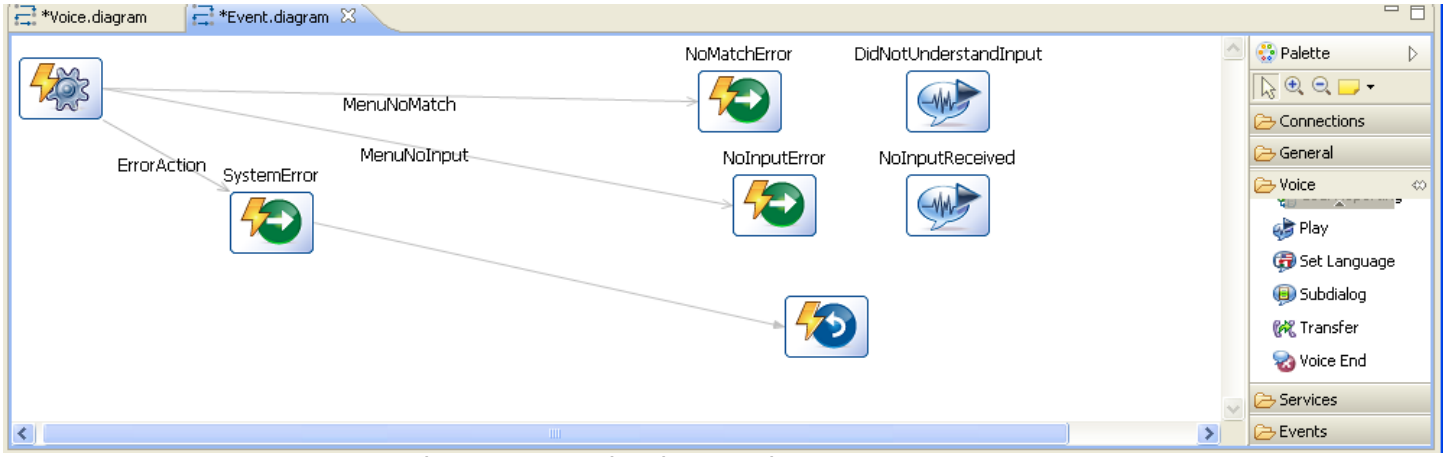


Figure 6.17 - Updated Event Diagram

After adding the Play components, connect them to the appropriate NoMatchError and NoInputError. Since we got here because of a fairly serious error issue, at this point it is good practice to transfer the caller to someone that can help. A subsequent recipe (Basic Transfer recipe) will discuss the call transfer in more detail. For this recipe, incorporate Play and Transfer components into the event diagram (Fig 6.18).

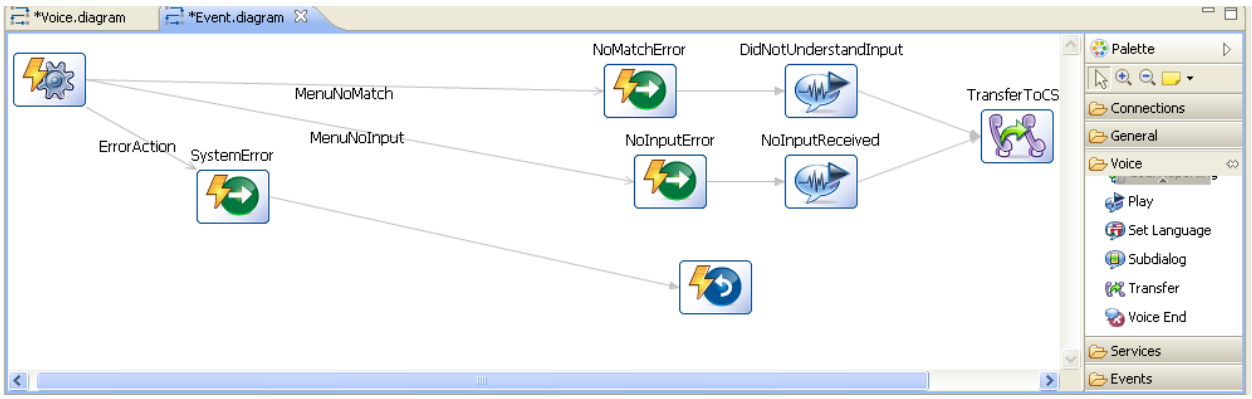


Figure 6.18 - Transfer Component Added to Event Diagram

The details of the Play and Transfer components will not be covered here. Refer to the **BasicTransfer** recipe for more information about transferring to an operator. If you are trying this recipe in a test environment that doesn't have transfer capability or a handy live operator, you can leave out the Transfer component. Just play the appropriate error message, and end the call.

Each Play component will contain a specific message to instruct the caller of what is going on. For DidNotUnderstandInput Play component, the system plays, 'Sorry I did not understand your input. Now I will transfer you to someone who can help.'

For NoInputReceived Play component, the system plays, 'Sorry I still did not receive any input. I am going to transfer you to someone who can help.' See Figure 6.18.

Note: Remember to connect the **Event Entry** component with the **Play** component. Add in a **Transfer** component and make the connections from the **Play** component to the **Transfer** component. See Figure 6.18.

The last error action for this sample recipe is the ErrorAction event. By default, the ErrorAction connects to a default Event Entry followed by a connection to an Event Return. Delete the **Event Return** and have the Event Entry play an Error message and connect to the Transfer component. This is the path that will be followed when the Error Event handler from the NewsWeatherSportMenu component is processed. This is the path that is taken for system errors. See Figure 6.19.

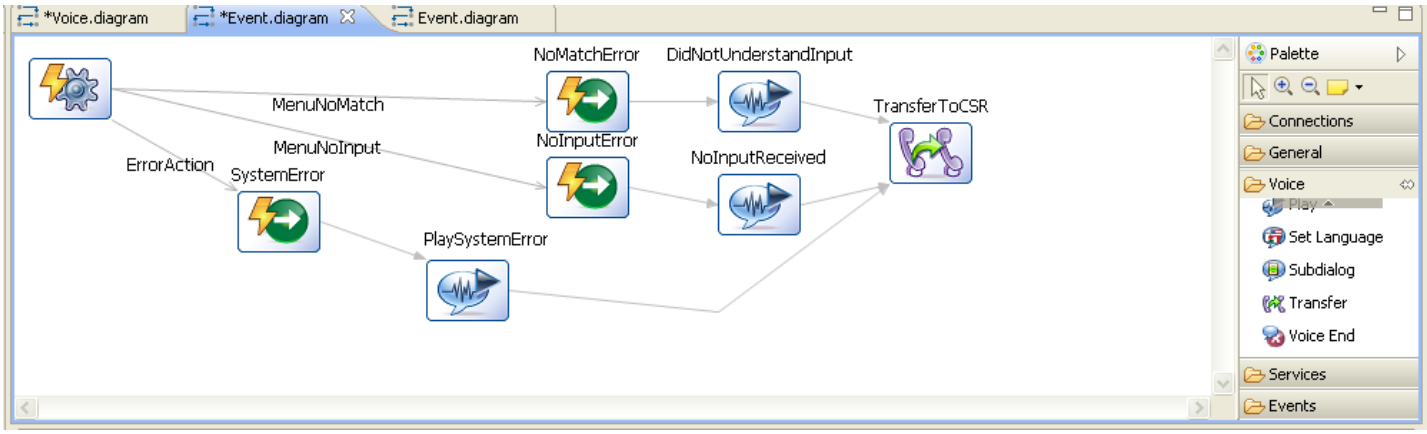


Figure 6.19 - Completed Transfer Component Added to the Event Diagram

Finally, go to the **Palette > Voice** drawer and add a **Voice End** component after the Transfer component. Connect the **TransferToCSR** component to it. See Figure 6.20.
Note: IC requires this component to clean up the call and reset the state of the application. The Transfer component doesn't take care of this automatically, even though it has effectively transferred the call from the application. With the completion of the Event diagram, we have successfully modified the NewsWeatherSportsMenu to support error recovery.

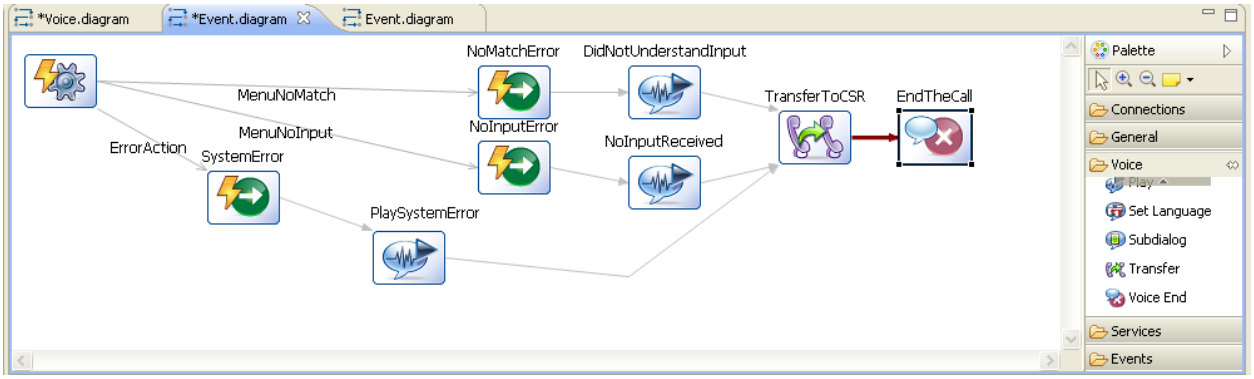


Figure 6.20 - Added Voice End Component to the Event Diagram

Final Notes for Application Completion

After all components are attached, proceed to **Package Explorer > applications >call**, and complete the **Call.diagram**. See **Recipe 1 – Hello World**. Remember that the call control diagram is required for the application to operate correctly in a runtime environment. Now, the application is complete and ready to be packaged to create the war file. See **Recipe 1 – Hello World**. After you complete the package export and create the war file, you are ready to put your application into runtime. You have now been successful in creating your Error Recovery NewsWeatherSports application project. Congratulations!

Recipe 7: Best Practices Error Recovery for Speech NewsWeatherSports

Contributed by Melissa Tatum.

Approximate time: 50 minutes, depending on prior experience

What you will learn with this recipe:

- The Types of Errors That Occur in a Speech Menu.
- How to use the Collect Component With Error Recovery
- How to define the number of errors allowed when processing the Collect component.
- Understand where the Error prompts are defined in the Collect component.
- Understand the Event diagram.

What the Error Recovery for SpeechNewsWeatherSports Application Does

As we stated in the previous recipe, the menu is one of the most important parts of a dialog system. Recipe 3's SpeechNewsWeatherSports recipe showed us how to present a caller with a spoken menu of choices, using IC's Collect component which is the speech equivalent of the DTMF Menu component. With the Collect component, the caller will make a menu selection by speaking one of the menu choices presented in the dialog.

Just like our first DTMF Menu recipe, the first SpeechNWS recipe wasn't very user-friendly. If the caller made a mistake by speaking an invalid selection or waited too long to speak a choice, that caller was dumped out of the application with nothing but a "goodbye".

Now we need to make our SpeechNewsWeatherSports speech menu just as user-friendly as we made our DTMF Menu. The same philosophy about retries, escalating prompts, error counts, and centralized event handling applies to speech menus just as it does in keypad menus. We want to make sure those users who don't enunciate clearly enough, or are hesitant to proclaim their choice, will have more than one chance to get their selection right. See the first few sections of the Recipe 6 on DTMF error recovery.

As with the DTMF Menu, we will try to show what we consider best practices in dealing with errors that callers often make when selecting menu choices using speech. This recipe will focus on using the Collect component to implement those best practice error recovery schemes.

We have already discussed much about error counters and event handlers in the previous recipe, so we won't go into that much detail here. We will assume that you understand the basics of the event handler, and will just touch on the differences from the DTMF Menu component error handling. Our goal will be to come up with a user-friendly Speech menu dialog structure that you can reuse in any application.

What does Error Recovery Actually Mean?

There are three speech errors that can occur in the Collect component – one more than what can occur in a DTMF Menu component. Two of the speech menu errors are the same as in a DTMF Menu - NoMatch, and NoEntry. The meanings of these errors are the same as in a touchtone menu, except the user input is speech, not a keypad. If the user says an invalid selection, a NoMatch error is generated. If the caller doesn't speak a response to the prompt within the time allotted, a NoEntry error is generated.

However, a speech menu has one additional error condition that DTMF menus don't have. The new error condition is called SpeechTimeout. This error occurs if the caller talks too long in response to the menu prompt. A DTMF Menu can't have a "Talk too long" error, since pressing a valid key will make the selection, even if the key is held down long after the selection is triggered.

Developing the Call Flow

Tip: Close all diagrams, and collapse the folders in Package Explorer.

1. Create a new project, and name it **ERSpeechNewsWeatherSports**. Create a **Voice.diagram**.
2. Open the **SpeechNWS** recipe project (Recipe # 3)
3. Open the recipe's **SpeechNWS Voice.diagram**
4. Select the following. See Figure 7.1 (the selected components have a black box around them):
   - **SNWS Menu**
   - **Case** component SNWS
   - Four **Play** Components (**News**, **Weather**, **Sports**, and **Goodbye**)
   - Do **not** copy the lower Error Log component.
5. Right-click the mouse on one of the selected components and select **Edit > Copy**.
6. Open the newly created **ERSpeechNewsWeatherSports** application and paste the components into the Voice diagram (see Figure 7.2).
7. Reconnect the **Voice Begin** and **Voice End** components as shown below.

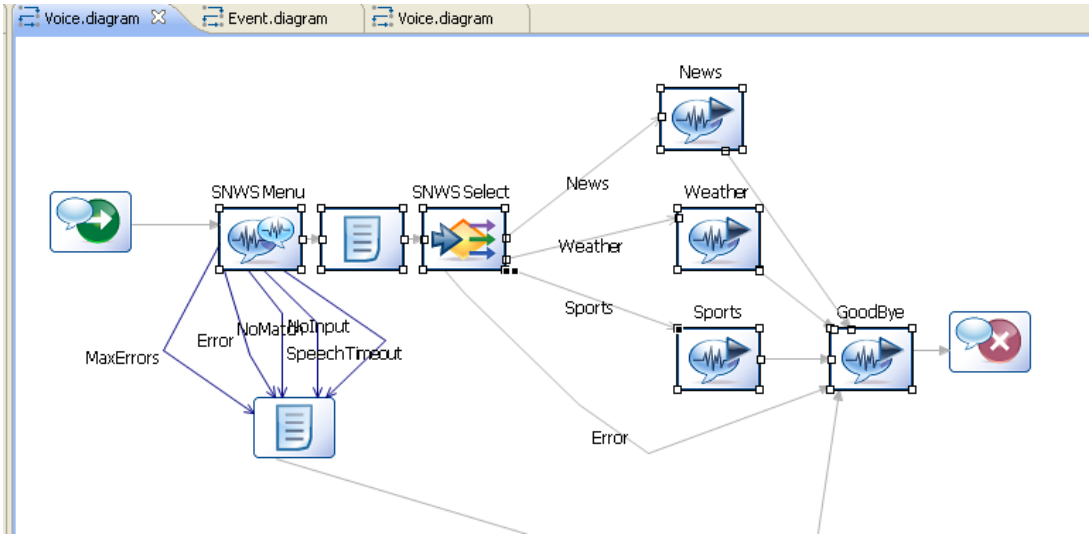


Figure 7.1 – Copying Components from Speech NewsWeatherSports Voice Diagram

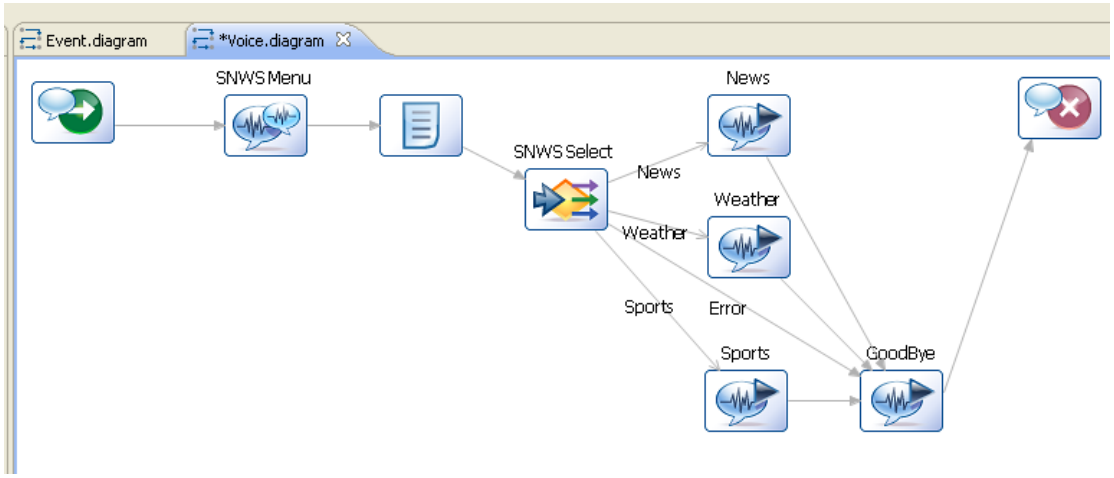


Figure 7.2 - Pasted Components into ERSpeechNewsWeatherSports Voice Diagram

Defining the Error Event Messages in SNWSMenu

The next step involves defining the specific error messages for our error recovery dialog. We want to specify unique messages for the NoMatch, NoInput and MaxSpeechTimeout errors. We also want to escalate the error messages if the caller makes a second error of the same kind, just like in the DTMF Menu case.

The error messages will be played as long as the number of errors encountered is less that the max number of Errors defined (see Figure 7.3). We will set the maximum number of errors to 3, just like the DTMF Menu.

Select the **Collect** component, and go to **Properties** view > **General** option > **Prompt** tab. Here is where we will discover that adding prompts in a Speech menu is slightly different than in the DTMF Menu. You should see that there is no Event subtab under the Prompts tab. All of the prompts, the original default menu prompt and the error prompts, are all in the one Prompts tab pane.

The original main or default prompt "Would you like the news, weather, or sports?" should already be in the prompt box. We defined this in the original SNWSMenu, and then copied all the components and their properties from the original recipe. The default Prompt Group holds our initial prompt, which plays the options for the caller. If you select the default Prompt Group, you will see that the Scope attribute has been set to **Default**. This is the Scope you should always use for the standard prompt in a Collect component, as it will cause that prompt to be played as soon as the Collect component is executed.

In the Prompt Group area, right-click in a blank space, and select **Insert a new prompt group below** (see Figure 7.3). The new Prompt Group will be automatically labeled Default_1, until we change its attributes.

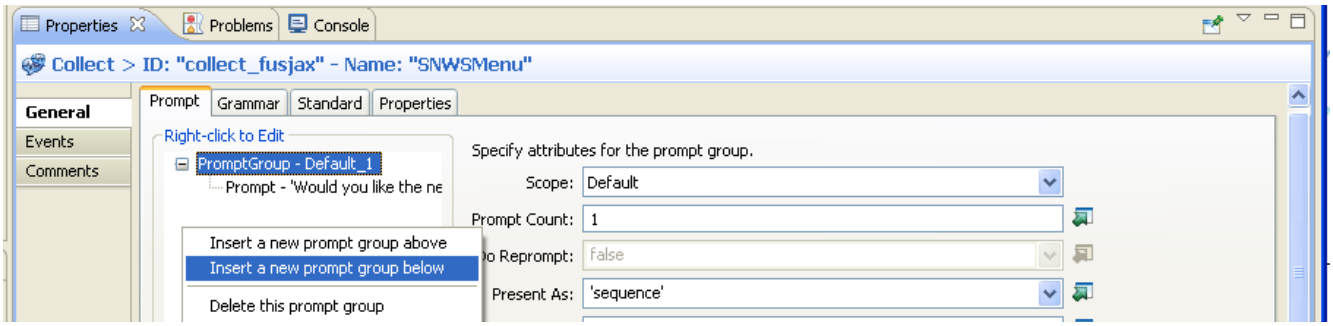


Figure 7.3 - Adding an Event Prompt Group

Select the new **Prompt Group**. Its attributes display in the right pane. Just as in the DTMF Menu, this is where you select the type of error (Scope) for which you want this prompt played, and when in the escalating prompt sequence it is to be played. In the Scope attribute, select **NoInput**. That way, this new prompt will play on the first occurrence of a NoInput error. For the Prompt Count, accept the default **1**. See Figure 7.4.

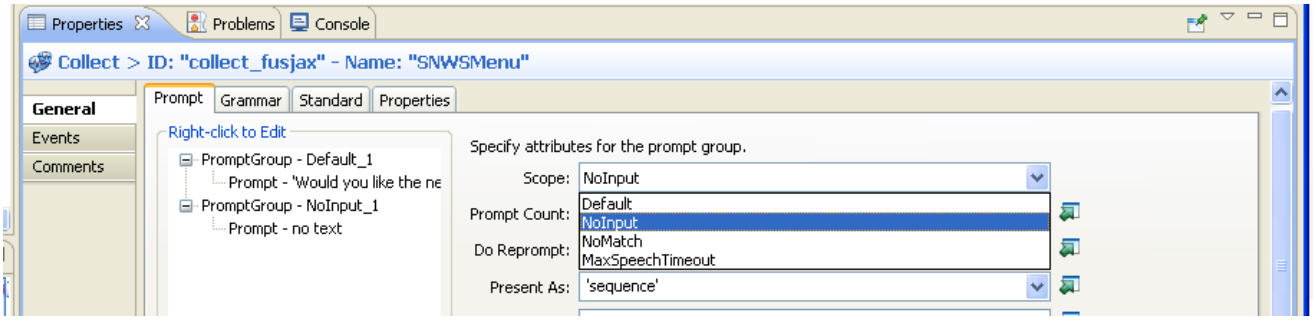


Figure 7.4 - Selecting Scope for Prompt Group

After we have changed the scope and prompt count of a prompt group, the name of the prompt group will change. As in the DTMF Menu, the Prompt Group name is defined by a combination of the scope followed by

the prompt count. The Scope is set to **NoInput** and the Prompt Count is defined to be **1** so the name of that prompt group is **NoInput_1**. (See Figure 7.5). Set the Do Reprompt field to **true** from the dropdown box. This means that we will play the Default_1 prompt again for the caller if the first NoInput error occurs. This is helpful because it reminds the caller what their choices are.

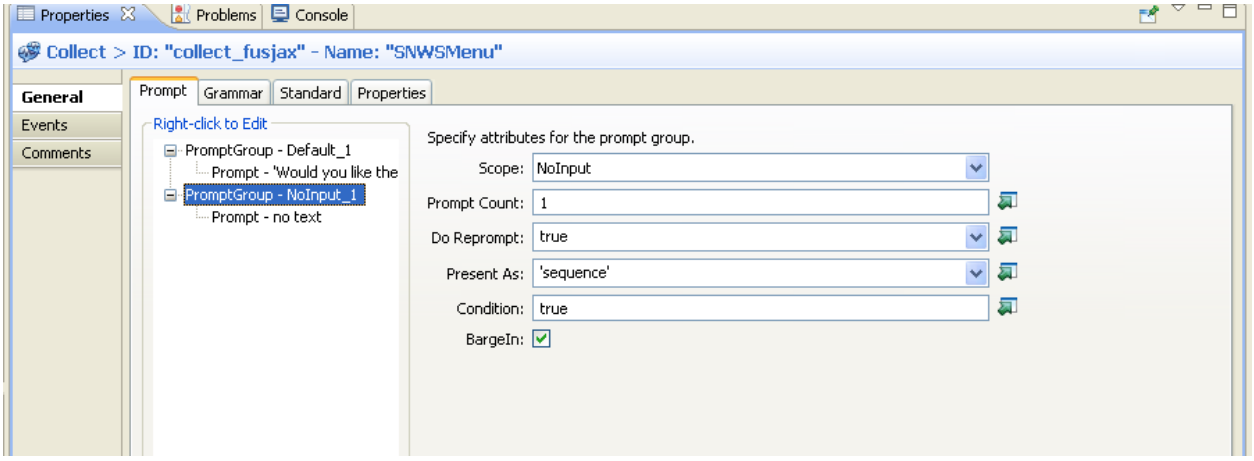


Figure 7.5 - NoInput_1 Prompt Group

Now that the NoInput_1 Prompt Group has been defined, let's define the specific prompts. Notice below NoInput_1 Prompt Group, we have Prompt – no text defined. Highlight **Prompt – no text** and you will be able to define the specific prompt to be played when NoInput_1 is encountered. (See Figure 7.6).

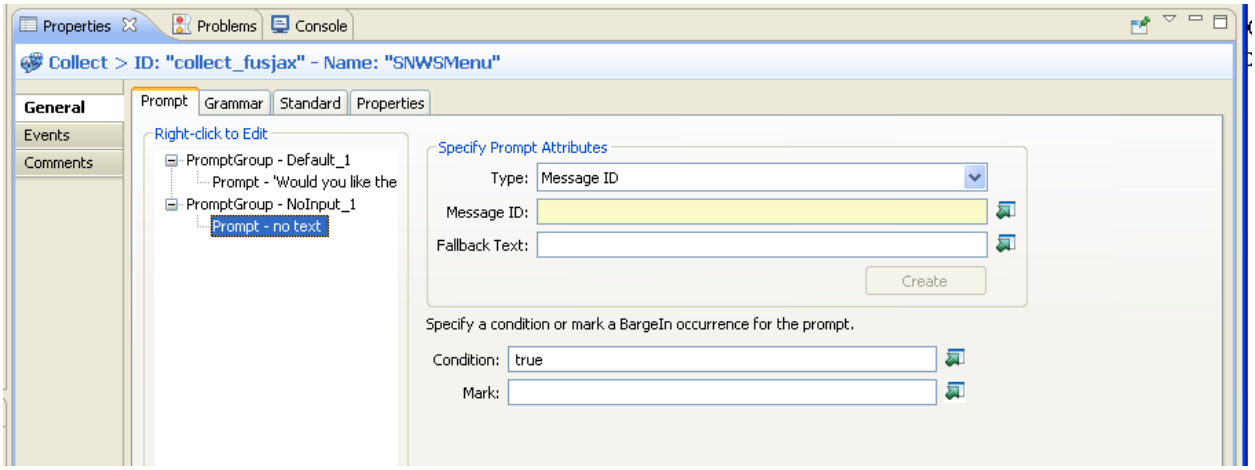


Figure 7.6 - Defining a New Prompt

Just as we defined prompts on previous recipes, highlight the **Prompt - no text** under Prompt Group – NoInput_1 and define the message to be played. In the Type field, select **TTS**. For the Expression, use single quotes and type, 'Sorry I did not get that. Please speak your selection.' See Figure 7.7.

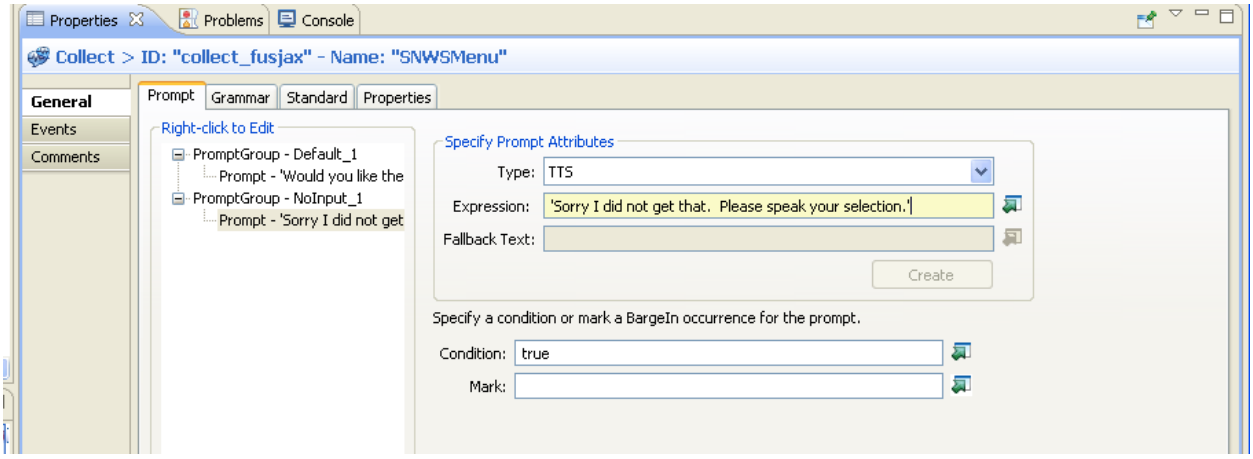


Figure 7.7 - Completing the New Prompt

Add a second **NoInput** error Prompt Group. Then add two escalating **NoMatch** prompt groups to the Prompt pane. Finally, add two **Speech Timeout** prompt groups to the pane. Speech Timeouts are our new type of error, and they occur when the caller talks too long when making a menu selection. This kind of error can't happen with a DTMF menu.

Remember that we only need two Prompt Groups for each error type, since we are going to set the maximum error counts for all types of errors to three. So, on the third error we will jump out of the Collect component and go to the Event Handlers which we will define later in this recipe. The previous DTMF Menu recipe covers the rationale behind the event handlers in more detail. Also, remember that with the second occurrence of each error type, the prompt messages get more specific.

Make sure that you set the appropriate **Scopes**, **Prompt Counts**, and **Do Reprompt** values for each Prompt Group. Your completed Prompt Groups will should be similar to Figure 7.8.

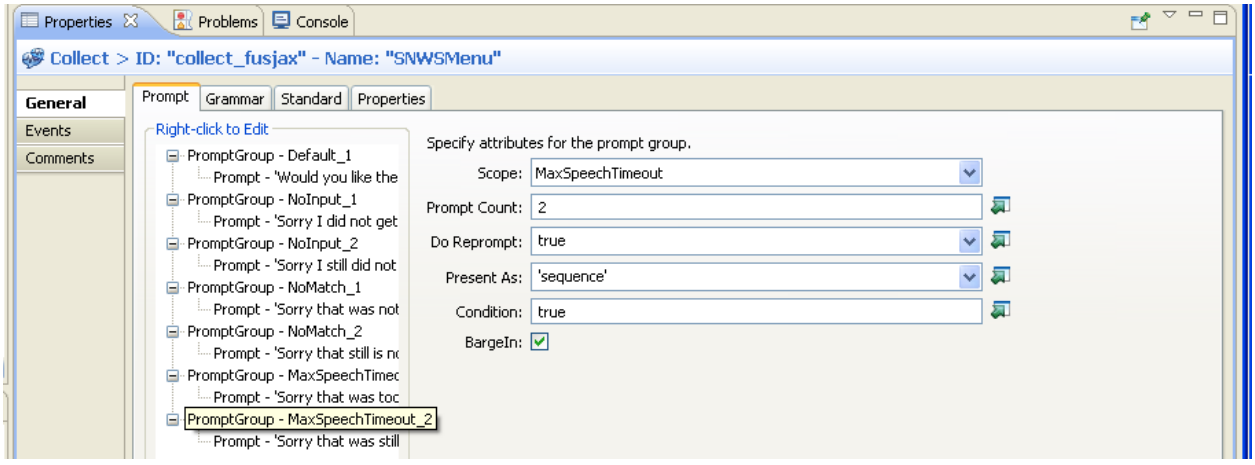


Figure 7.8 - Completed Prompt Groups and Messages for NoInput, NoMatch, and MaxSpeechTimeout

If you have seven prompt groups in your Prompt pane, you have completed the error prompt configurations. The first time the caller speaks an invalid choice, Prompt Group –NoMatch_1 will be played. The second time the caller speaks an invalid choice, Prompt Group-NoMatch_2 will be played. The third time the caller speaks an invalid choice, the error counter will have reached its maximum count. At this point processing will go to the event handler.

The Event Handler

We covered the functionality and rationale for the Event Handler in some detail in Recipe 6, so we won't say much more about it here. Suffice it to say that we will be configuring the Collect component in our recipe to route the call flow to the event handler when any maximum error count is reached in any of the error counters in the Collect component. This is best practice for all menus in an application, which allows the handling of critical errors to be located in one place in the application. This simplifies maintenance and reduces clutter in your diagrams.

The Collection Components' Maximum Error Counters

The NoInput and NoMatch errors that callers make in Speech menus are very similar to the DTMF menu errors of the same name. However, there is the one additional error condition for speech - the Speech Timeout error. Each error type has an error counter of the same name in the Collection component, so there are three speech error counters in the Collect component. Along with those three error counters, there are two other error counters, one for system errors (Error) and one that counts any kind of user input error – NoInput, NoMatch, or SpeechTimeout (MaxErrors). So we have a total of five error counters in the Collect component:

MaxNoMatch—This counter increments each time the caller speaks a selection that is not a valid menu option. If the caller says "Music" in the Collection component menu for example, that will increment the NoMatch error count.

MaxNoInput—This counter increments each time the caller fails to speak their selection before the entry timer times out. If a caller doesn't say anything at all after the menu prompt, that event will increment the NoInput counter.

MaxSpeechTimeout—This counter increments each time a caller talks longer than allowed, after the menu prompt. If a caller just keeps talking after the menu prompt, this event will increment the SpeechTimeout counter.

MaxErrors—This counter counts whenever a NoMatch, NoInput, or Speech Timeout error occurs. This essentially keeps the sum of the three user error types. For best practices we typically don't want to allow the caller to make more than three errors of any type. After that, we want to go to the event handler. This counter makes that kind of behavior easy to set up.

The time settings for the various "didn't talk", and "talked too long" parameters are defined on the Properties view of the Collect component. The specifics of these are not covered in this recipe.

Error—This counter counts errors that are not NoMatch, NoInput, or Speech Timeout errors. These are typically system or programming errors. For example, a TTS error, or a missing audio prompt file, will count as this type of error. These kinds of errors are usually catastrophic, since it means that the system can't play a specific prompt. You won't get to set a maximum error count for this condition. The max count for this condition is forced to **one**, and you can't change it. You will be able to select an event handler for this condition, however.

Specifying the Event Handlers for Each Max Error Condition

The next step in completing our speech menu error recovery recipe is to define what specific event handler will be called when a specific error counter reaches its maximum. You should remember how we did this in our previous recipe, as this will be much the same process, except there is one additional error counter we need to configure.

Select the **Collect** component and click on the **Events** tab. You will see that there are five counters - Error, MaxNoMatch, MaxNoInput, MaxSpeechTimeout and MaxErrors. These error counters are used to keep track of how many errors a user makes while in the Collect component. As before, you can use the dropdown box to specify how you would like to proceed if one of the counters exceeds its maximum count.

For each error counter we can choose the event handler that will be executed if that counter hits its max level. We discussed the rationale behind event handlers in the previous recipe, so you should understand that the selections in the dropdown boxes of each counter simply represent entry points into the global event handler which we will define in the next sections.

Use the dropdown box to select the event handler for each error counter. Best practices dictate that the following event handlers should be chosen for each counter:

For Error select **Error**
For MaxNoMatch select **Collection Error**
For MaxNoInput select **Collection Timeout**
For MaxSpeechTimeout select **Collection Error**
For MaxErrors select **Collection Error**

Figure 7.9 shows the appropriate event handlers set for each error counter.

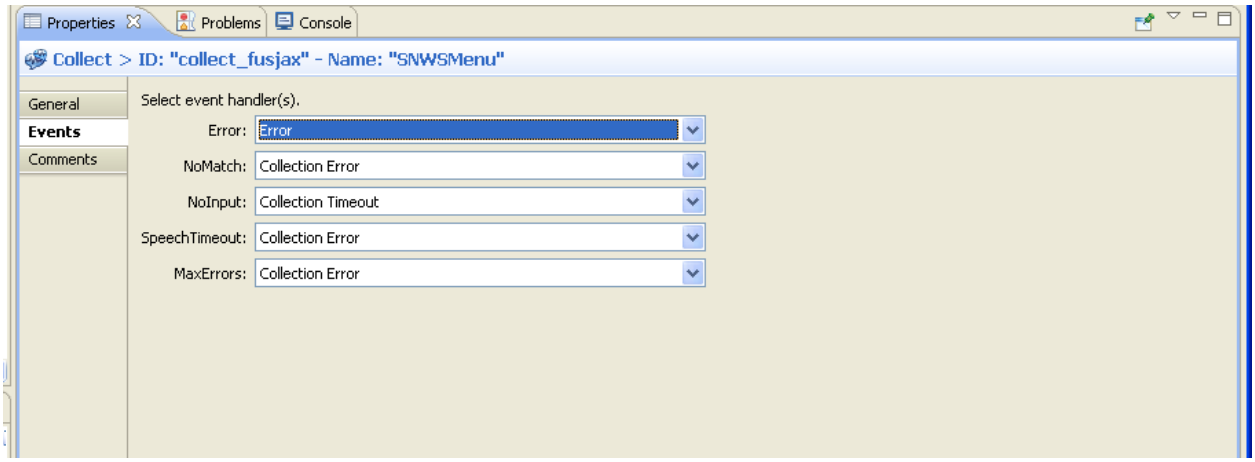


Figure 7.9 - Selecting the Event Handlers for the Errors

Setting the Maximum Error Retry Limits

We still need to define the number of errors that we want to allow the caller, before we bail them out of the Collect component altogether, to get more help. This configuration process is almost identical to what we did in the DTMF Menu Error recipe, except with speech, we do have one additional error type – the MaxSpeechTimeout error.

With best practices, the Max Speech Timeout max count error is just handled with the same event handler that does error handling for Max No Match and Max Errors. This makes some sense, since reaching any of these three max error counts (NoMatch, SpeechTimeout, or MaxError) means that the caller has provided some input, but just couldn't get it right. Even after three tries. Only the NoInput counter represents a different condition, since a max count condition on the NoInput counter means that the caller never made any input attempts at all, after three presentations of the menu prompt. The caller's phone microphone may be on mute, or some other issue. This condition is often treated differently than the other caller input max error conditions.

On the SNWS Collect Menu component, go to **Properties** view > **General** option > **Standard** tab. On this tab you will see where you can set the four maximum count limits for the four counters: Max Errors, Max No Input Max No Match, and Max Speech Timeout. Remember that the fifth counter, called the Error counter, counts system errors, and it's max count is not configurable, as it is always set to **one**; therefore, Error is not in this list. Figure 7.10 shows the Standard tab settings after we have completed setting all of the fields.

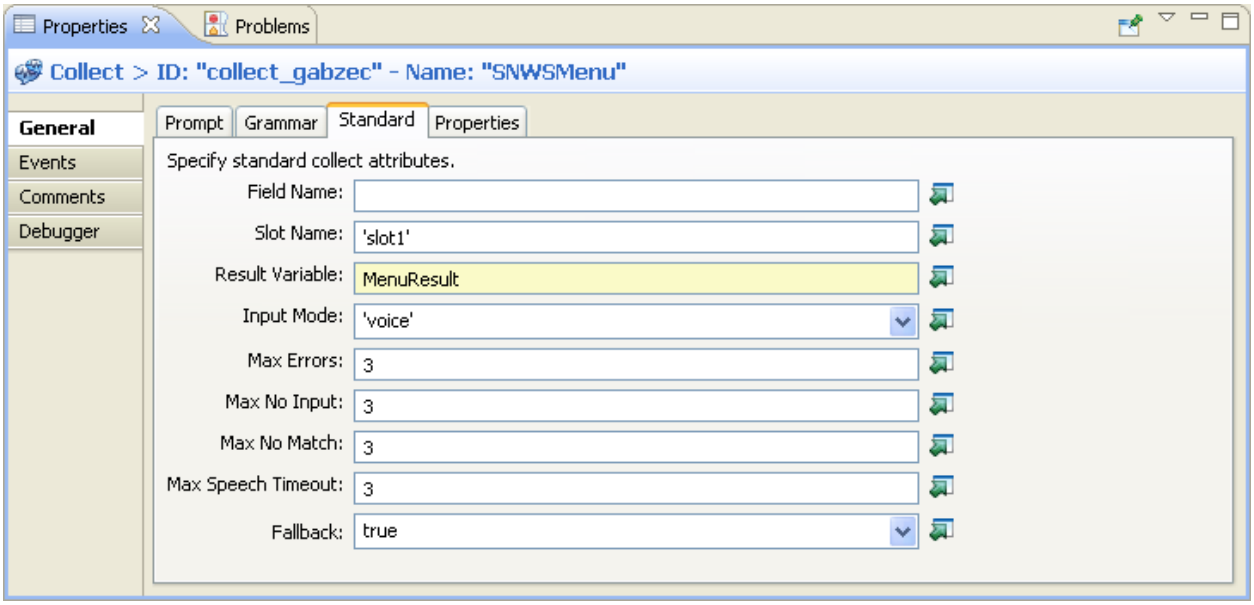


Figure 7.10 - Defining the Maximum Number of Errors

Setting the counter maximums and event handler associations for each counter completes the work we need to do in the Collect component for this recipe. The remaining work will be done in the Event diagram, which will let us define the details of what happens if we encounter a maximum error condition.

Completing the Event Diagram

From our previous error recovery recipe, you should remember that the Event.diagram is located under the **applications > voice** folder. This Event.diagram will be configured in about the same way as in the last recipe, except that we will use two new entry points or event handlers, which are different than the DTMF Menu error event handlers we used previously.

One would think that we wouldn't treat speech menu errors any differently than touchtone entry errors. Max errors in DTMF and speech menus have very similar meanings. However, there are some subtle differences, such as the details of what is said in the error prompts– I didn't HEAR any input, vs I didn't RECEIVE any input, etc. We won't necessarily take advantage of these subtleties in this recipe, but the extra entry points make sure that we could.

Launch the default **Event.diagram** by double-clicking on the file in the **voice** folder. This process follows the same steps as in the previous chapter so we won't go into as much detail here. (See Figure 7.11).

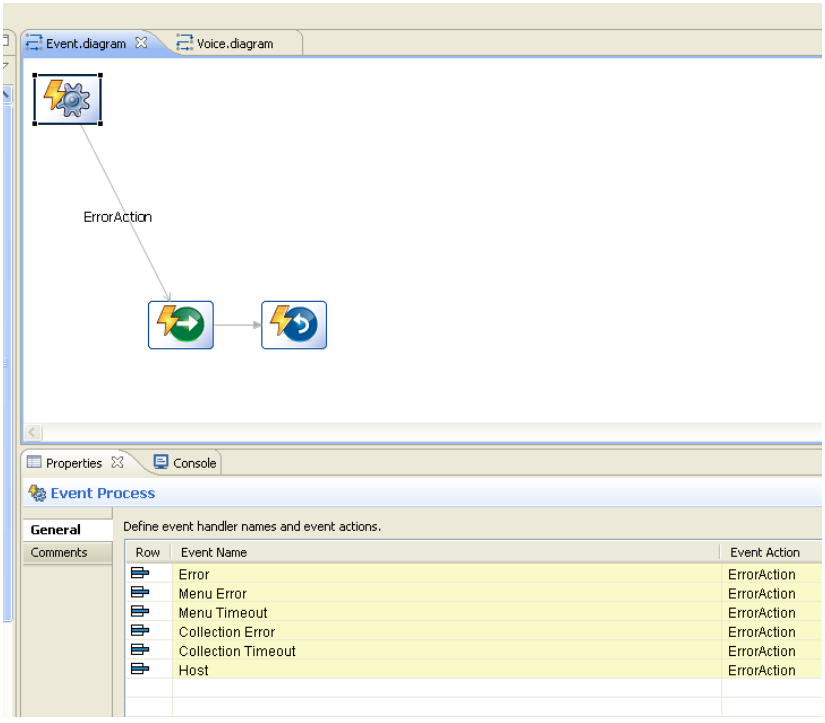


Figure 7.11 - Default Event Process in the Event Diagram

From the Events drawer, add two new **Event Entry** components into the **Event.diagram**. These will be to handle the new Collection Error and Collection Timeout error conditions. When we configured the Collect component's error counters, we lumped all of the errors that indicate some kind of user action into one category – the Collection Error. We also set the Collect component such that if we never get any input from the user, we will call the Collection Timeout error. Figure 7.12 shows this step.

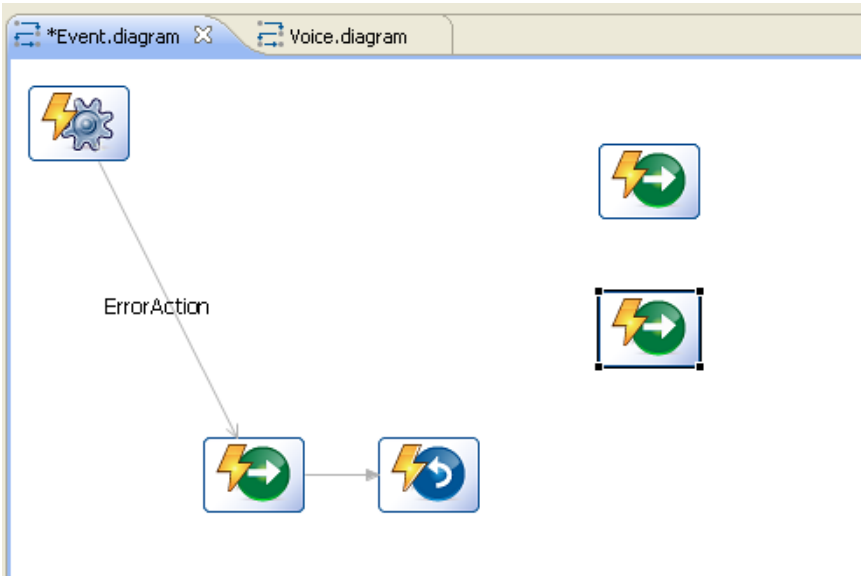


Figure 7.12 - Adding Event Entry Components to Event Diagram

Configuring the New Event Entry Components

Now that we have added in the two new Event Entry components, we can integrate them into the Event.diagram. Select the **Event Process** component in our diagram. In the Properties view, go to the Event Name's **Collection Error** and **Collection Timeout** entries and configure the **Event Action** properties. Set the Collection Error event action to **CollectionNoMatch** and the Collection Timeout event action to **CollectionTimeout**. See Figure 7.13.

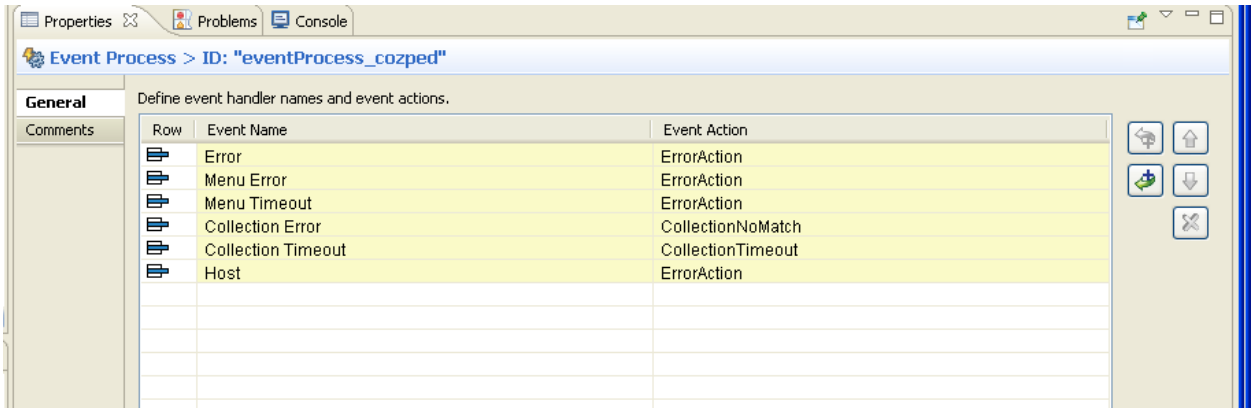


Figure 7.13 - Event Process Component with Customized Errors Actions

Now we can connect the Event Process component to the Event Entry components, and select the appropriate connector for each type of error path. Figure 7.14 shows the path selection mechanism, and Figure 7.15 shows the completed connections

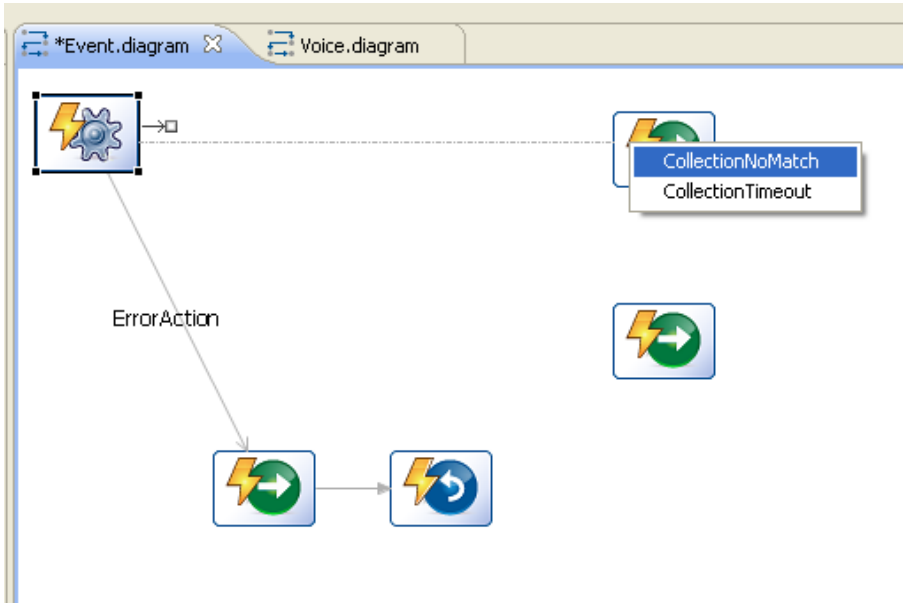


Figure 7.14 - Connecting Event Entry Components to Event Process

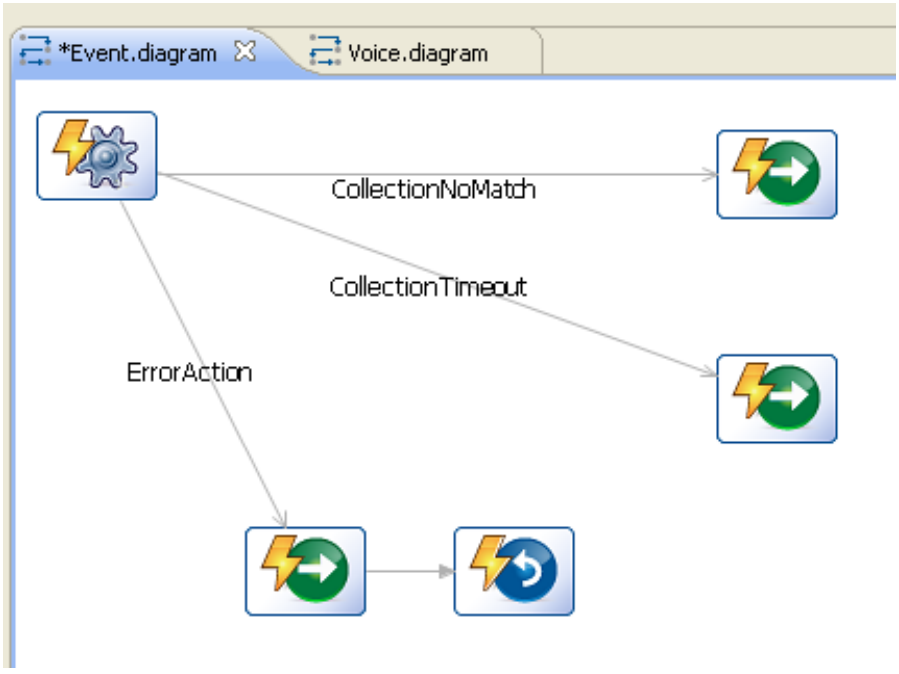


Figure 7.15 - The Completed Event Entry Connections

Now add the **Play** components for each error path that will notify the caller of the specific problem that was encountered, and what we are going to do about it. Label the NoMatch Play component **DidNotUnderstandInput**, and the NoInput Play component **NoInputReceived**. Remember that you can label a component on the diagram by pressing F2 or by selecting that component with the mouse and start typing.

Configure the **DidNotUnderstandInput** Play component to play 'Sorry I did not understand your input. Now I will transfer you to someone who can help'. Configure the **NoInputReceived** Play component to play: 'Sorry I still did not receive any input. I am going to transfer you to someone who can help'. Remember to connect each Play component with the appropriate Event Entry Component. See Figure 7.16.

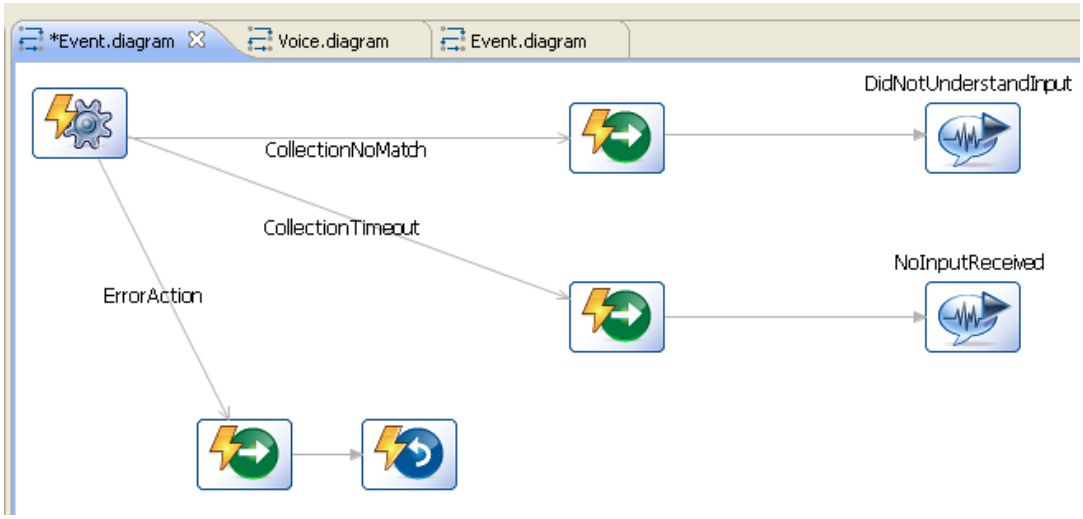


Figure 7.16 - The Updated Event Diagram

Now, from the Voice drawer, add a **Transfer** component. Make the connections from the **Play** components to the **Transfer** component. See Figure 7.17.

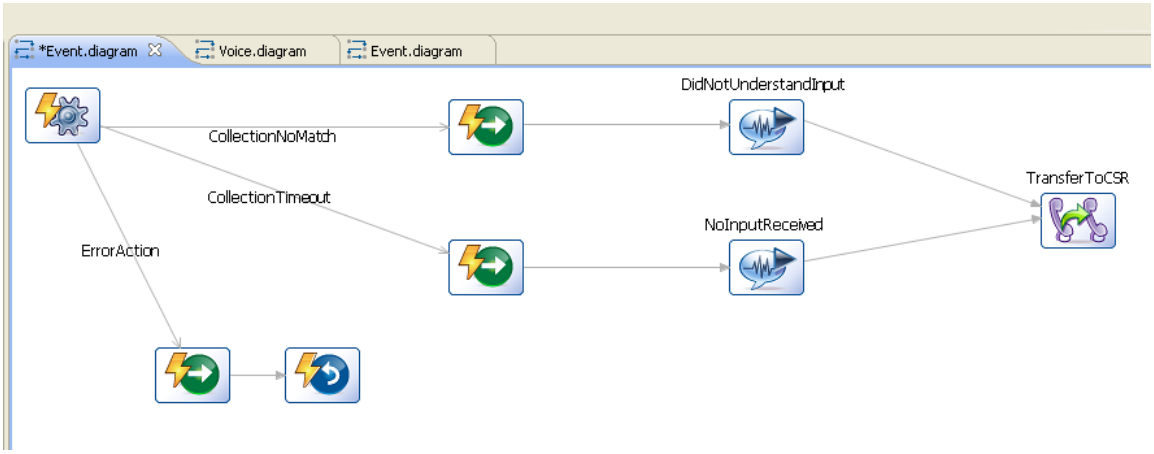


Figure 7.17 - Transfer Component Added to the Event Diagram

We still need to define what happens on the basic Error Action event, which is the default path taken when the Collect component encounters a system error condition. Get rid of the Event Return component, and add a third **Play** component for the third error path. The Play prompt should play 'There has been an error with the system. I am going to transfer you to someone who can help.' Then add the **Voice End** component and make sure everything is connected up. See Figure 7.18.

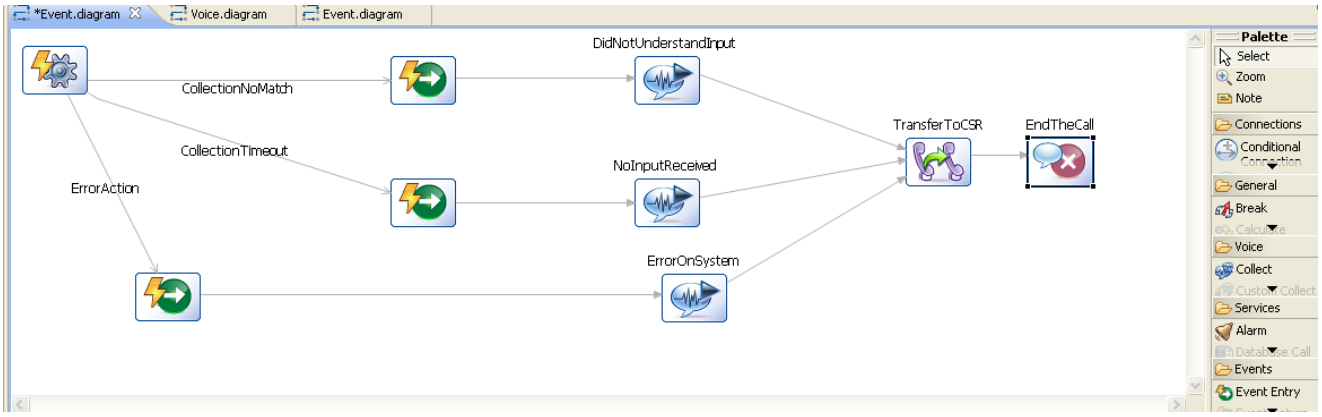


Figure 7.18 - Final Event Diagram

Final Notes for Application Completion

After all components are attached, go to Package Explorer and complete the Call diagram under **applications > call**. Save everything, export the package and create the War file. After you complete the package export and create the war file, you are ready to try out your error-recovered Speech Menu recipe. You should keep this recipe around, as you will find it very handy, whenever you build any application that has a Speech menu in it. Great job!

Recipe 8: Call Transfer

Contributed by Melissa Tatum.

Approximate time: 30 minutes, depending on prior experience

What you will learn with this recipe:

- Understand the Meaning of Transfer
- Understand the Various Types of Transfer
- Understand the Calculate Component
- Understand the Transfer Component

What the Basic Transfer Recipe Does

In previous recipes we showed you how to use the Transfer component to transfer a caller to an operator if they were having trouble. In this recipe we will use the Transfer component a bit differently. We will be doing transfers, but it won't be because the caller has made too many mistakes. This recipe sets up a fictional bank called "Your Hometown Bank" with three departments:

- New Accounts
- Loan
- Customer Service

Our recipe will allow the caller to choose which department they want to transfer to, essentially automating the job of a receptionist. This type of application is often called an "automated attendant", and is quite common in many retail businesses. This recipe introduces the Calculate component as well as the Transfer component. We also make use of the Play, Collect, and Case components which have been covered in previous recipes.

What does Transfer Actually Mean?

Transfer means that the caller is going to be connected with another party. The party can be a call center agent, customer service representative, or even another voice response application. For example, imagine that you want to get information on opening a new bank account. You simply call the phone number associated with the bank's voice response application. You will then most likely hear a greeting followed by a list of options. If you select the "open new bank account" option, the voice response system simply connects you with the bank's New Accounts customer service representative. With this connection, you have just implemented a transfer!

What are the Various Types of Transfer?

This section explains the three basic types of transfers: blind, consultation, and bridge.

Blind transfer—This is the simplest type of transfer. The IVR system places a call to a new number, and the caller is transferred to that new line. At the same time, the IVR application that made the transfer is completely disconnected from the call. Making a blind transfer usually assumes that the application is finished dealing with the calling party, and has handed off the caller to be dealt with by a live agent, or another IVR application. If the application needs to get a transferred caller back if the transfer is unsuccessful, or if the transferred call has completed, we would use one of the other types of transfer.

A blind transfer doesn't give the IVR application any information about what happened with the transferred call. A blind-transferred call could ring with no answer, get a busy signal, get a wrong number announcement, or even get a call rejection, but the transferring IVR application will not know what happened. Only the transferred party (original caller) will know what happened with the transfer, since they will be listening to the call's progress. Blind transfers are best used when transferring to a line that you know will be answered, either by a live person or another IVR system.

A blind transfer typically signals the end of any dealings that the application will have with the calling party. The IVR application normally ends immediately after a blind transfer step. There are ways to do wrap-up after a blind transfer (in the event handler routines), but since the original caller has been transferred to another line, there isn't anyone for our IVR application to talk to. Voice operations (Play, Collect) after a blind transfer are useless. Database clean-up or other maintenance operations are about all that can be done at this point, and they will have to be done in the event handler routines.

The Blind transfer has a second type of transfer that is for VoIP only. It is known as SIP (Session Initiation Protocol) transfer. Again, after the call is connected to another party, we have no information available about the status of the call transfer.

Consultation transfer —This is basically an extension of the blind transfer. Remember that a blind transfer didn't provide us any information about the status of the call transfer, and it didn't give us any options, even if we did know what happened to the call. Once a blind transfer is made, the IVR application is "out of the loop" and the call is essentially ended, as far as the IVR application is concerned.

A Consultation transfer gives us some information about what happened after the transfer, and better yet, it allows us to do something about it. A consultation transfer will let us know about whether the caller was successfully connected with the called party or not. If the caller was not successfully connected, the IVR system can take back the transferred call, and give the caller other options which we can define in our voice response application. However, if the caller is successfully connected to the transferred party, then the consultation transfer will end the IVR system's participation in the call, just as the blind transfer does.

A Consultation transfer should be used when we aren't sure whether the transferred number is good or not. This allows us to give the original calling party other options, if the transfer isn't successful.

Bridge transfer—This transfer keeps our IVR application "in the loop" even after a transfer has been successfully completed. It lets the application monitor a transferred call, and "take back" that call if it hasn't ended (caller hung up).

The most important feature of the bridged call transfer is the fact that after the transfer has been made, the IVR system can still "listen" to the original calling parties' audio on the transferred call. The IVR won't recognize the audio from the transferred (called) parties' line, but it can recognize if the original caller presses a DTMF key, or speaks a "hot word". Thus if an application requires that a caller have the option to return to the original IVR application after a transfer has been made, use the bridged transfer.

A bridge transfer also reports information about the call transfer process, including whether the called party was busy, or didn't answer. In addition a bridge transfer will report how a transferred call ends, indicating whether the caller or the called party (perhaps a customer service representative) hung up the phone.

Use a bridged call transfer if we want to give a transferred caller the option to return to the original IVR application, after a called party (transferred-to party) hangs up. Alternatively, we can allow the calling party the option to return to the original IVR application anytime during the transferred call, under the callers control, by using a DTMF key or some other asynchronous event.

For this sample recipe, we will be using the simple Blind transfer.

What is the Purpose of the Calculate Component?

The Calculate component is used to evaluate expressions, and assign values to variables, that are used in a call flow. The expressions are defined in ECMAscript. In this recipe, we are using the Calculate component to simply assign department phone numbers to specific variable names, and those variable names will be used in the Transfer component. With that being said, let's get started on our call flow!

Developing Call Flow

Start by opening IC and creating a new **ICS Application Project** called **BasicTransfer** followed by the creation of an **ICS Voice Diagram**. For more specific information about creating the project and voice diagram, refer to **Recipe 1**.

Adding Play, Collect and Case Components to Voice Diagram

For specifics about adding and configuring the Play, Collect, and Case components, see **Recipe 3 (Speech NewsWeatherSports)**. Create the flow in the following order:

- **Play** component labeled **PlayGreeting**.
  - **TTS**
  - Expression: 'Welcome to Your Home Town Bank.'
- **Collect** component labeled **SpeechMainMenu**.
  - **TTS**
  - Expression: 'What would you like to do? Please say new account information, loans, or customer service representative.'
- **Case** component, labeled **CheckMainMenuSelection**, where we check the spoken response from the caller at the SpeechMainMenu. The Case component will branch based on the caller's spoken input.
  - Expression: **MenuResult.result**
  - Else Label: **default**
  - Result Equals (3 rows): **'NewAcctInfo', 'Loans', 'CustomerServiceRep'**
  - Then Label (3 rows): **NewAcctInfo, Loans, CustomerServiceRep**

Now, our updated Voice diagram will be displayed as the following (see figure 8.1).

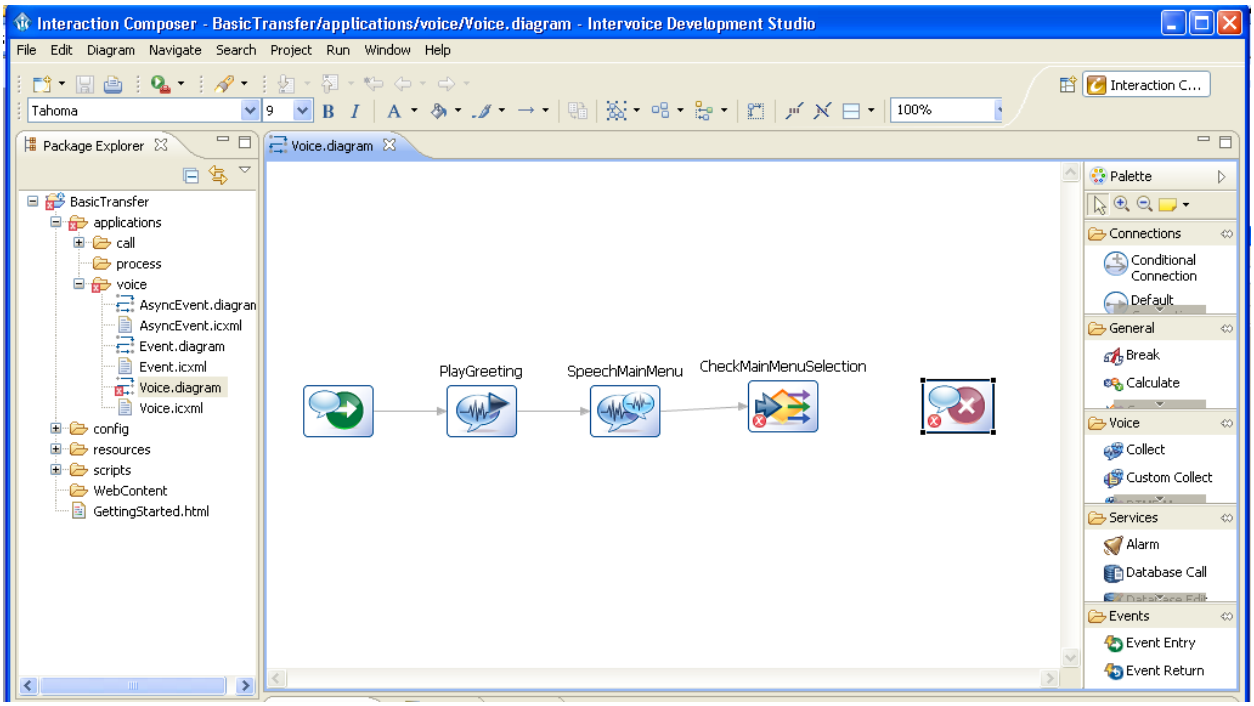


Figure 8.1 - Updated Voice Diagram

Now that we have added the base components to our Voice diagram, we are ready to explore the new components and ideas that this recipe introduces. Our next step is to add Calculate components to the Voice diagram.

Adding Calculate component to Voice Diagram

A Calculate component is used to calculate expressions and assign values to variables. From the Palette's General drawer, add three **Calculate** components to our Voice diagram, placing the Calculate components to the right of the Case component. We want each of the three Calculate components to assign a different value (phone number) to the same variable name, depending on which branch is taken from the Case component.

Label the three Calculate components **GetNewAcctInfoExtension**, **GetLoanExtension**, and **GetCustServiceRepExtension** respectively. Select the component, and start typing your label.

Note: If you need to edit it an existing label, you can select the component and press the F2 key, which will put the label in edit mode.

After adding and labeling the three Calculate components, our Voice diagram should look like figure 8.2.

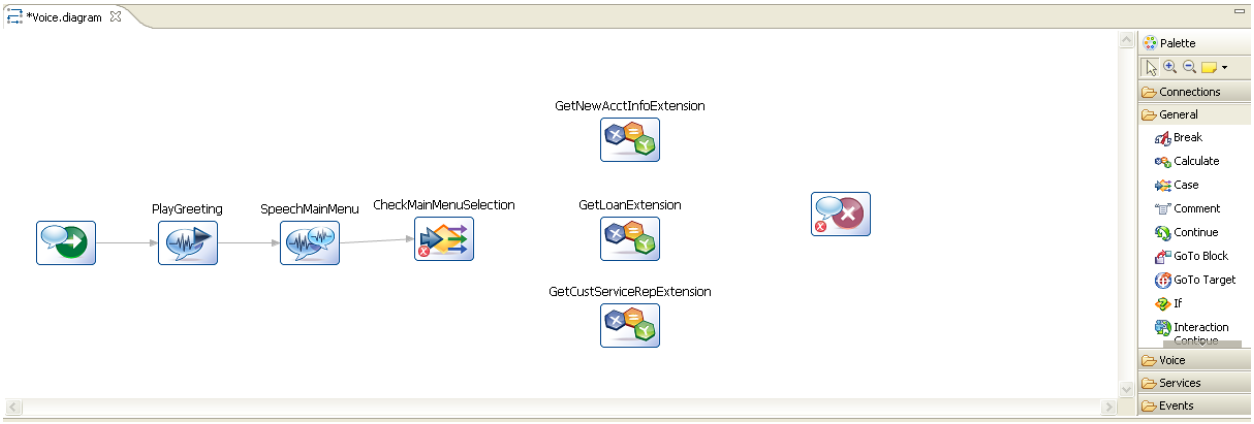


Figure 8.2 - Adding the Calculate Components

Defining the Calculate Components' Properties

We need to define a result variable in this recipe that will carry the extension number that will be passed to the Transfer component to execute the transfer. This will be the result variable in all three Calculate components.

Select the first **Calculate** component (**GetNewAcctInfoExtension**) and type the result variable name in the Calculate component's **Properties** view > **General** option > **Result** field as **transferExtension**. If this first Collect component is executed, the transferExtension variable will be assigned the extension number of the New Account Information department.

In the Expression field, type the phone extension value. For this recipe, we have defined the integer **6113131** as the extension number for the New Accounts department. In a production application, the extensions to various departments should be obtained from the bank's telephone directory. See Figure 8.3.

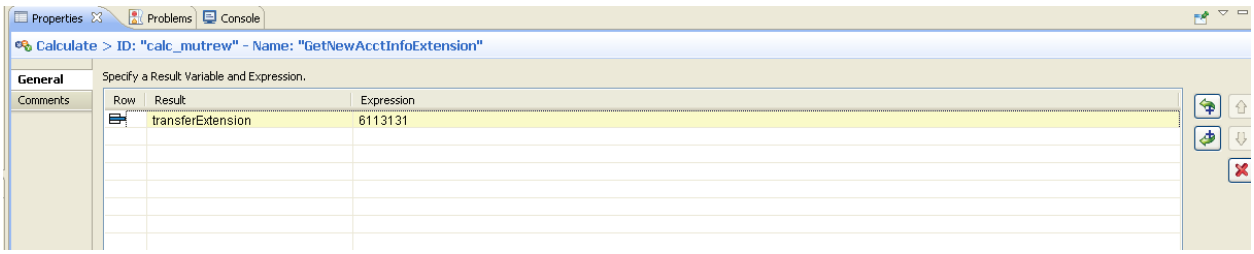


Figure 8.3 - Completing GetNewAcctInfoExtension Calculate Component

We will continue with the same process to define the second Calculate component. This time we are assigning **6118964** to the transferExtension variable, which is the Loan departments' extension (see Figure 8.4). Note that this is a different extension number than the previous Calculate component. Depending on which Calculate component is executed coming out of the Case component, the transferExtension variable will be assigned to different extension numbers.

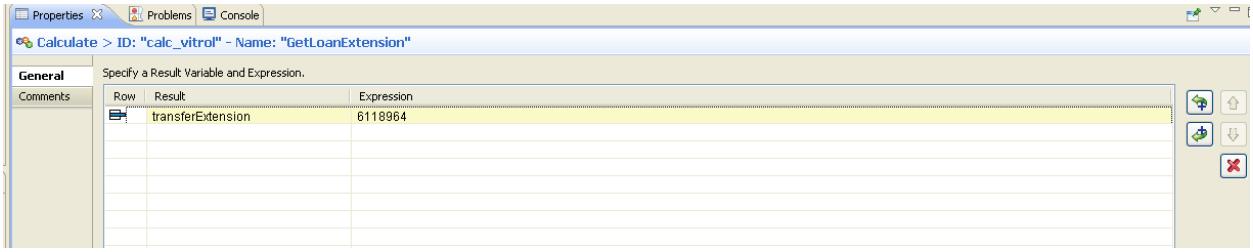


Figure 8.4 - Completing GetLoanExtension Calculate Component

Finally, complete the third Calculate component labeled **GetCustServiceRepExtension**. Here just as in the previous two Calculate components, assign the customer service representatives extension to transferExtension. This time we assign **6113333** to transferExtension.

Now all of the Calculate components are complete. At this point, CheckMainMenuSelection Case component can attach to the designated Calculate component (again refer to **Recipe 3 SpeechNewsWeatherSports** for information on connecting the case component to other components). The updated Voice diagram with the connections from the Case component to the Calculate components is now in place (see Figure 8.5).

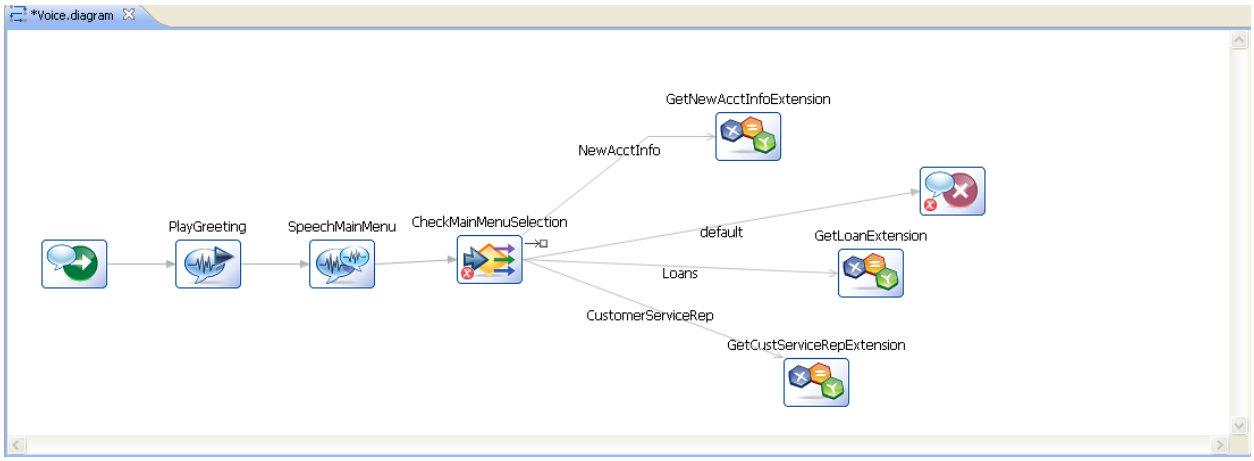


Figure 8.5 - Updated Voice Diagram with Calculate Components

Adding a Transfer Component to the Voice Diagram

Finally, from the Palette's **Voice** drawer, add a **Transfer** component to the Voice diagram (see figure 8.6).

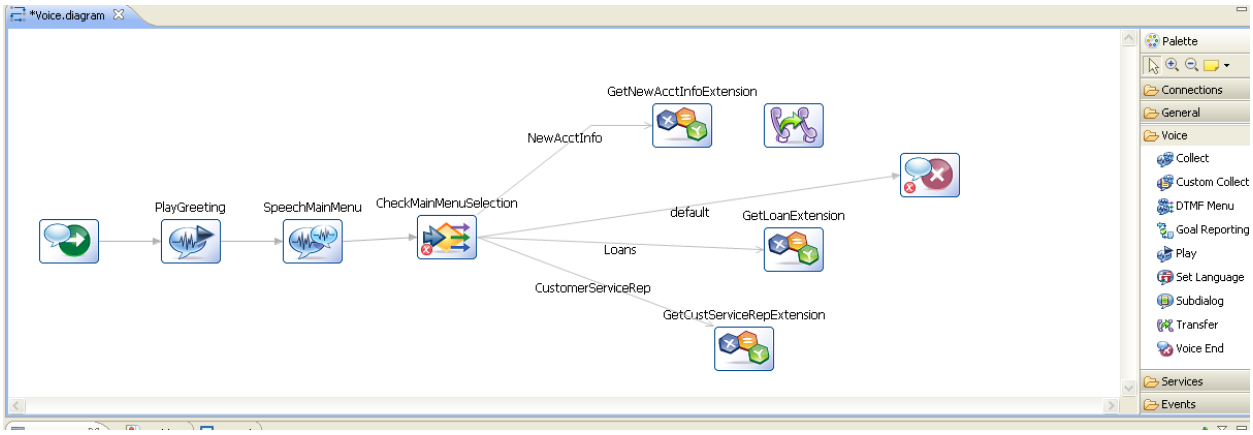


Figure 8.6 - Adding in Transfer Component

For this sample project, label the component TransferCall. In this case, we are trying to make the labels as descriptive as possible (See figure 8.7)

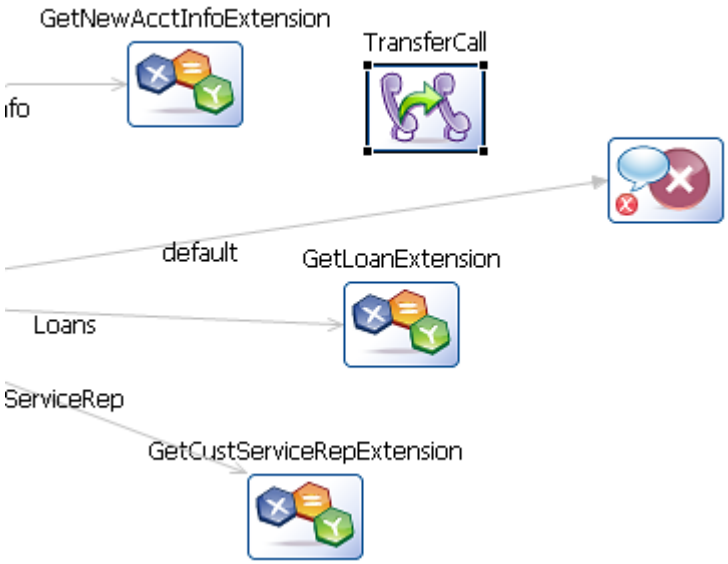


Figure 8.7 - Labeling Transfer Component

Completing the Transfer Component

Select the **Transfer** component, and examine the Properties view. There are several parameters for the Transfer component that can be manipulated. One helpful parameter is the Transfer Audio parameter. With this parameter, an audio message can be played during the transfer. However, for this particular recipe, we are only concerned with the Transfer Type and the Destination Expression. Notice, that for Transfer Type, you choose a value from the dropdown box. In this case, we will choose **Blind** (this will designate the transfer as being a blind transfer discussed earlier in this recipe). For the Destination Expression, we will use the variable **transferExtension** that we assigned on the previous Calculate components. See Figure 8.8.

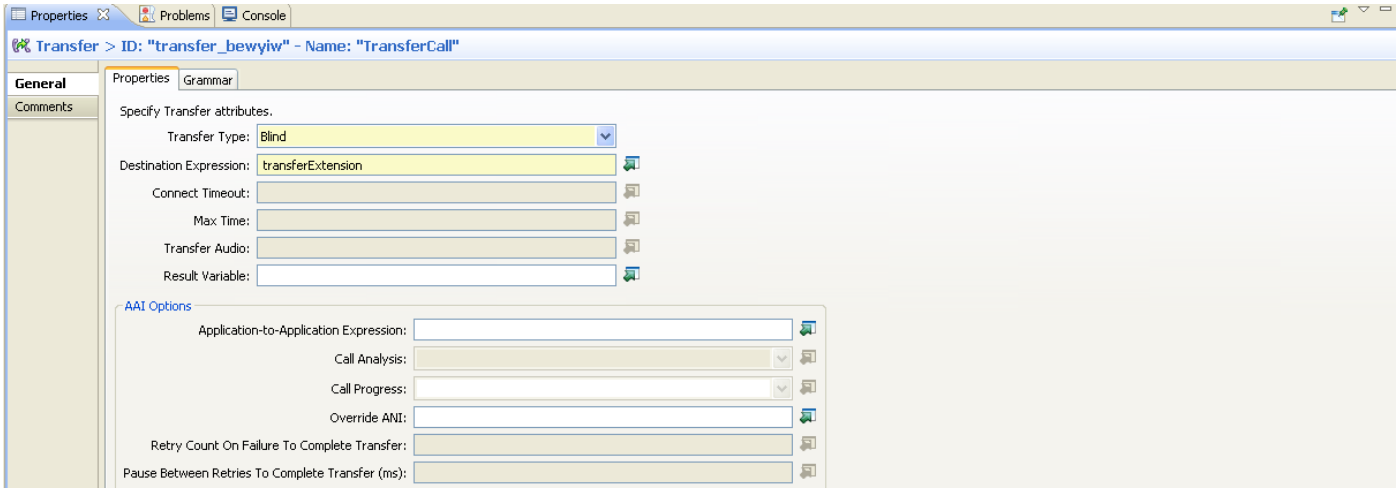


Figure 8.8 - Completed Transfer Component

Making the Final Connections to the Components

Since the Transfer component has now been completed, connect all the **Calculate** components to the **Transfer** Component.
Note: These are default connections. You don't need to add a label.
Finally, connect the **Transfer** component to the **Voice End** component. The completed Voice diagram will look similar to figure 8.9.

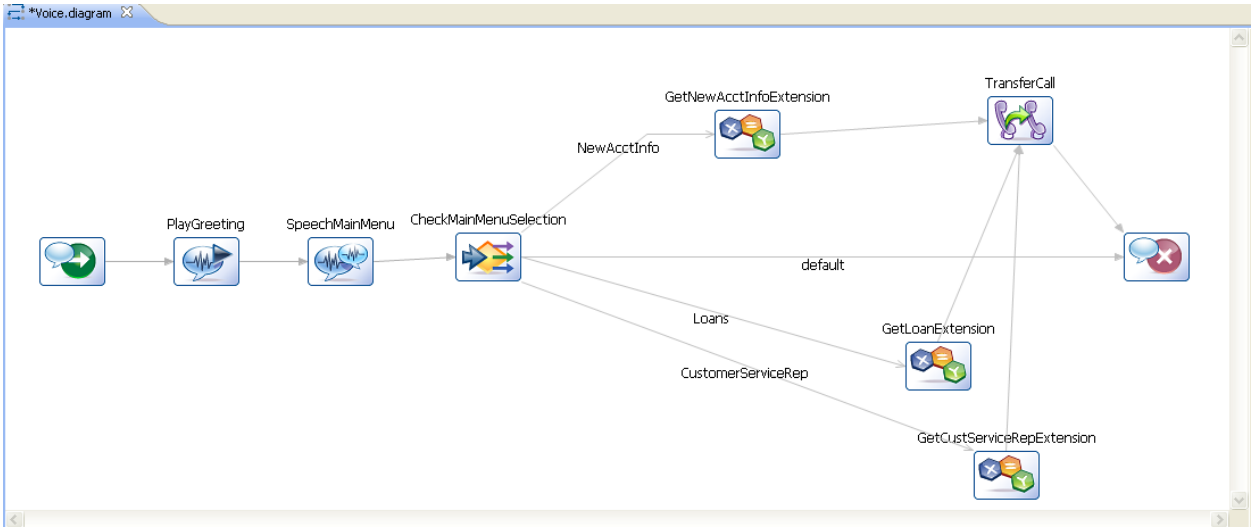


Figure 8.9 - Completed Voice Diagram

Final Notes for Application Completion

Once all components are attached, complete the Call.diagram under **applications >call**. See **Recipe 1 – Hello World**. This is needed for the application to be able to work in a runtime environment.

Now, the application is complete and ready to be packaged to create the war file. See **Recipe 1 – Hello World**.

After you complete the package export and create the war file, you are ready to put your application into runtime. You have now been successful in creating your Basic Transfer Application. Congratulations!

Technical support

If you have a technical question or problem related to a Convergys product, refer to your service contract and/or support guidelines for your Convergys representative contact information.

If you have a technical question or problem related to a Convergys product, check your service contract and/or support guidelines to find your regional support office location and contact information.

You can access RealCare by:

1. RealCare Customer Self-Service Portal: http://support.convergys.com/irj/portal/ (You must register to obtain a user name and password.)
2. Email address: RealCare.TS@convergys.com
3. Mailing address: 17787 Waterview Parkway Attn. RealCare, Dallas TX 75252

Otherwise for assistance, contact technical support in one of these geographical areas:

- Western hemisphere
- Eastern hemisphere

Western hemisphere

For 24-hour support, contact the following regional support office:

	National dialing	International dialing
USA	Tel: (800) 955-4688 Tel: (972) 454-8130	Tel: +1 800 955-4688 or Tel: +1 972 454-8130

Eastern hemisphere

During normal office hours, contact one of the following regional support offices:

	National dialing	International dialing
England	Tel: 0161 495 1234 Fax: 0161 495 1007	Tel: +44 161 495 1234 Fax: +44 161 495 1007
Germany	Tel: 0611 184 440 Fax: 0611 184 4444	Tel: +49 611 184 440 Fax: +49 611 184 4444
Singapore	Tel: 6395 4300 Fax: 6395 4277	Tel: +65 6395 4300 Fax: +65 6395 4277
United Arab Emirates	Tel: 04 429 0602 Fax: 04 429 0601	Tel: +971 4 429 0602 Fax: +971 4 429 0601

After-hours support calls

If your contract includes after-hours support, contact:

	National dialing	International dialing
UK	Tel: 0208 750 6387	Tel: +44 208 750 6387

Note: After-hours calls are directed to a support desk, where you can record your problem details. You will be contacted as soon as possible.

relationship management