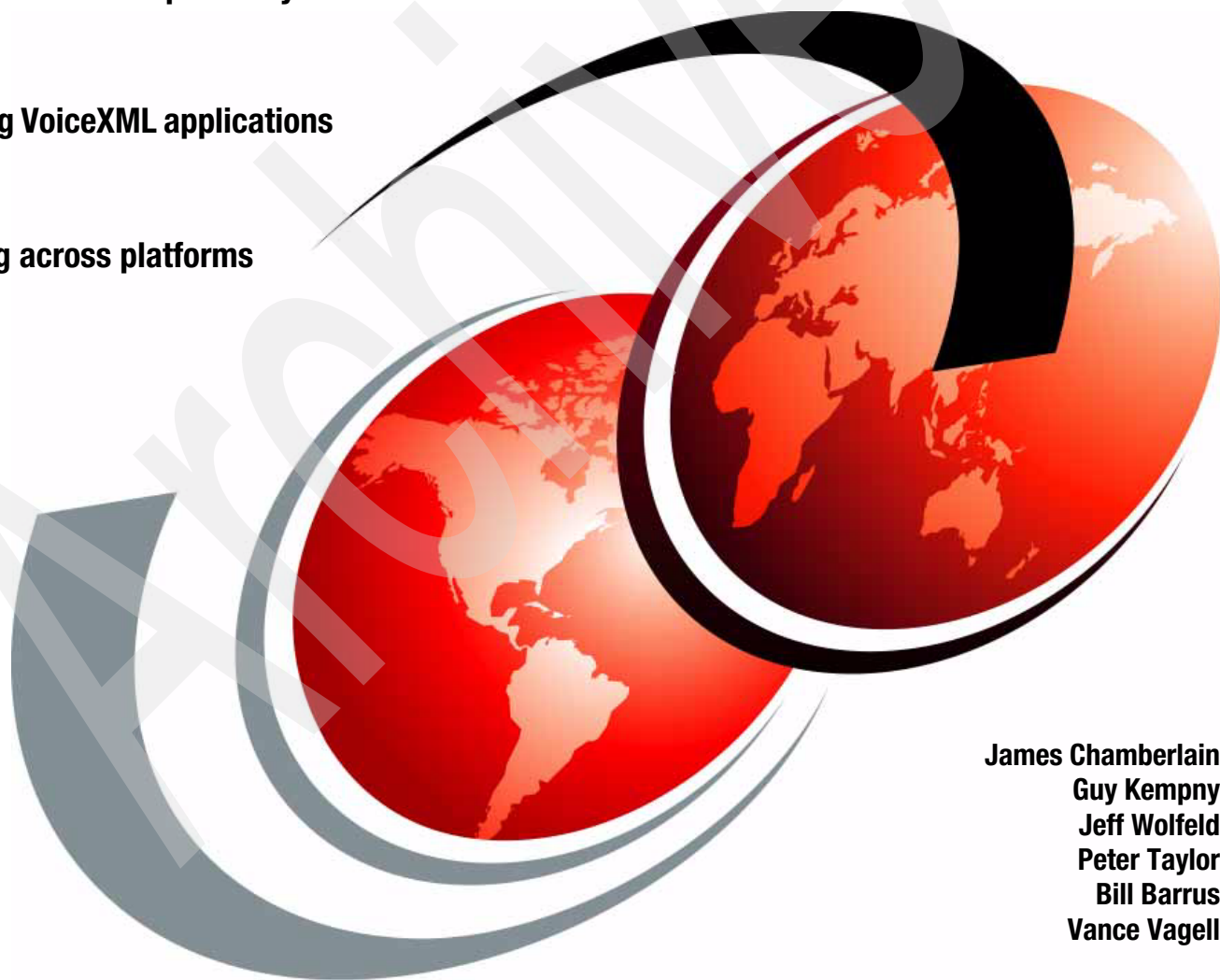


# IBM WebSphere Voice Server V5.1.2/V5.1.3 and Cisco Customer Voice Portal V3.1: An Interoperability Guide

Using MRCP for interoperability

Developing VoiceXML applications

Debugging across platforms



James Chamberlain  
Guy Kempny  
Jeff Wolfeld  
Peter Taylor  
Bill Barrus  
Vance Vagell





International Technical Support Organization

**IBM WebSphere Voice Server V5.1.2/V5.1.3 and Cisco  
Customer Voice Portal V3.1: An Interoperability Guide**

May 2006

Archived

**Note:** Before using this information and the product it supports, read the information in “Notices” on page vii.

### **First Edition (May 2006)**

This edition applies to Version V5.1.2 and V5.1.3 of WebSphere Voice Server and to Version 3.1 of Cisco Customer Voice Portal.

**Note:** This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.



# Contents

<b>Notices</b> .....	vii
Trademarks .....	viii
<b>Preface</b> .....	ix
The team that wrote this Redpaper .....	ix
Become a published author .....	xi
Comments welcome .....	xii
<b>Chapter 1. Introduction</b> .....	1
1.1 Overview of WebSphere Voice Server for Multiplatforms V5.1.x. ....	2
1.1.1 WebSphere Voice Server V5.1.3 features .....	2
1.2 Overview of Cisco Customer Voice Portal V3.1 .....	4
1.2.1 Core capabilities .....	4
1.2.2 Highly flexible architecture .....	4
1.2.3 The Cisco CVP solution .....	5
1.3 Typical interactive voice response scenario .....	8
1.4 High level architecture design .....	9
1.5 Interoperability .....	9
1.6 What comes next? .....	10
1.7 WebSphere Voice Server for Multiplatforms V5.1.3 .....	10
1.8 WebSphere Voice Server for Multiplatforms V5.1.3 integration .....	12
1.8.1 Basic deployment environment for a development .....	12
1.8.2 Advanced deployment environment for production .....	12
<b>Chapter 2. Basic deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3</b> .....	13
2.1 Architecture of a basic deployment solution .....	14
2.1.1 Cisco CVP V3.1 and WebSphere Voice Server in a basic deployment infrastructure .....	14
2.1.2 Basic deployment solution topology .....	14
2.2 Cisco Customer Voice Portal VoiceXML server .....	15
2.2.1 Software .....	15
2.3 Installing Cisco Customer Voice Portal VoiceXML server .....	16
2.4 Installing WebSphere Voice Server V5.1.3 .....	37
2.4.1 Obtain the software from the IBM Software Access Catalog .....	38
2.4.2 Prepare the systems and network operating systems .....	39
2.4.3 Run the installation using LaunchPad .....	40
2.4.4 Verify the installation .....	44
2.4.5 Network infrastructure for basic deployment .....	46
<b>Chapter 3. Advanced deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3</b> .....	47
3.1 Complex deployment overview .....	48
3.1.1 Complex deployment solution topology .....	48
3.2 ICM Integrated Deployment .....	49
3.2.1 Installation of the Cisco CVP V3.1 Call Control Server .....	49
3.2.2 Initial configuration of the Cisco CVP V3.1 Call Control Server .....	56

3.2.3 Initial Gateway and Gatekeeper Configuration for ICM Integrated Deployments .....	66
3.2.4 ICM Configuration .....	72
3.3 Overview of the WebSphere Edge Server	
Load Balancer component V5.1.1.41 .....	91
3.3.1 Load Balancer with WebSphere Voice Server .....	92
3.3.2 Hardware and software prerequisites .....	92
3.3.3 Microsoft Windows Server 2003 installation .....	92
3.3.4 Configuring the Load Balancer system .....	93
3.3.5 Network infrastructure for complex deployment .....	98
<b>Chapter 4. Developing voice applications with Cisco CVP V3.1 Studio</b> .....	101
4.1 General introduction to Cisco CVP VoiceXML technology .....	102
4.1.1 How VoiceXML addresses limitations of traditional IVR technology .....	102
4.1.2 Challenges inherent In VoiceXML development .....	103
4.1.3 How Cisco CVP addresses VoiceXML development challenges .....	103
4.2 Overview of Cisco CVP Solution Architecture .....	103
4.2.1 Cisco CVP VoiceXML Studio .....	104
4.2.2 Cisco CVP VoiceXML Server .....	104
4.3 Elements in studio .....	105
4.3.1 Exit states .....	105
4.3.2 Element and session data .....	106
4.3.3 Customization .....	106
4.3.4 Overview of Elements .....	108
4.4 Installing Studio and Server .....	118
4.4.1 Full Installation .....	118
4.4.2 Cisco CVP VoiceXML Server V3.1 Only Installation .....	123
4.4.3 Cisco CVP VoiceXML Studio V3.1 Only Installation .....	124
4.5 First steps in VoiceXML development in studio .....	124
4.5.1 Creating the HelloWorld project .....	125
4.6 Integrating an application with ICM call routing .....	310
4.6.1 Example overview .....	310
4.6.2 WeatherReport VoiceXML Studio application .....	310
4.6.3 Configuring WeatherReport to run in standalone mode .....	313
4.6.4 WeatherReport ICM routing script .....	314
4.6.5 Configuring the Gateway and Gatekeeper .....	323
4.6.6 Executing the WeatherReport application .....	323
4.7 IBM WebSphere Voice Toolkit V6.0.1 .....	324
4.8 IBM WebSphere Voice Toolkit V6.0.1 enhanced features .....	325
4.8.1 Lexicon pronunciation editor .....	325
4.8.2 Creating an external grammar file .....	329
4.8.3 Reusable Dialog Components .....	329
<b>Chapter 5. Testing and troubleshooting</b> .....	333
5.1 Installation testing .....	334
5.1.1 Cisco CVP phone call test .....	334
5.1.2 Cisco CVP call test requesting WebSphere Voice Server ASR and TTS functions .....	336
5.1.3 Testing Load Balancer within Cisco deployment .....	337
5.1.4 Network traffic analyzing .....	339
5.2 Troubleshooting a Cisco Gateway problem .....	343
<b>Appendix A. Cisco Router Configurations</b> .....	347
A.1 Cisco 1751-V Modular Access Router Configuration .....	347

A.2 Cisco AS5400HPX Universal Gateway Configuration .....	351
A.3 Cisco 3660 Gatekeeper Configuration .....	363
<b>Appendix B. Additional material</b> .....	367
Locating the Web material .....	367
Using the Web material .....	367
System requirements for downloading the Web material .....	367
How to use the Web material .....	368
<b>Glossary</b> .....	369
<b>Abbreviations and acronyms</b> .....	373
<b>Related publications</b> .....	375
IBM Redbooks .....	375
Online resources .....	375
How to get IBM Redbooks .....	376
Help from IBM .....	376

Archived

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## **COPYRIGHT LICENSE:**


This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

# Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX 5L™  
AIX®  
CallFlow®  
DB2®  
@server®  
eServer®

Everyplace®  
IBM®  
Lotus®  
OS/390®  
PartnerWorld®  
Rational®

Redbooks (logo) ™  
Redbooks™  
WebSphere®  
Workplace Client Technology™  
Workplace™  
xSeries®

Intel, Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

Intel®

The following terms are trademarks of the Microsoft Corporation in the United States, other countries, or both:

Microsoft®

Windows®

eXchange, Enterprise JavaBeans, Java, JavaBeans, JSP, J2EE, J2SE, Sun, Sun Microsystems, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Java™  
Sun Microsystems™  
Sun™

J2SE™  
J2EE™  
JSP™

JavaBeans™  
Enterprise JavaBeans™  
eXchange™

The following terms are trademarks of other companies:

Audium®, Say It Smart®, Audium Builder™, Call Services™, and Element™ are trademarks or registered trademarks of Audium Corporation in the United States, other countries, or both.

Cisco is a trademark of Cisco Systems in the United States, other countries, or both.

Gordon Kapes is a trademark of Gordon Kapes, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Red Hat is a trademark of Red Hat in the United States, other countries, or both.

SUSE is a trademark of Novell in the United States, other countries, or both.

The following is a list of W3C terms claimed as a trademark or generic term by MIT, ERCIM, and/or Keio on behalf of the W3C:

- ▶ W3C®, World Wide Web Consortium (registered in numerous countries), HTML (generic), HyperText Markup Language
- ▶ HTML (generic), HyperText Markup Language
- ▶ HTTP (generic), Hypertext Transfer Protocol
- ▶ XML (generic), Extensible Markup Language
- ▶ XHTML (generic), The Extensible HyperText Markup Language
- ▶ XSL (generic), Extensible Stylesheet Language

Other company, product, or service names may be trademarks or service marks of others.

# Preface

The next generation of IBM® WebSphere® Voice Server for Multiplatforms V5.1.2 and V5.1.3 products leverage the power of IBM WebSphere Application Server, an IBM flagship product and leader in the industry. WebSphere Voice Server V5.1 is the first release of a novel, patent pending architecture for speech servers, based on IBM WebSphere Application Server. No other speech server in the industry can match the on-demand capabilities that WebSphere Voice Server is able to provide as a result of its WebSphere Application Server base. Add the power of the Java™ 2 Enterprise Edition (J2EE™) platform including the runtime, services, and container management, and the result is a powerful speech server that integrates into existing IT infrastructures. Having voice on the same network as visual and data applications greatly reduces the initial speech investment. It simplifies the management and achieves improved integration with customer content, which is a feat that no other speech vendor can match.

The Cisco Customer Voice Portal V3.1 software product delivers next generation voice processing that meets advanced call-routing and voice self-service needs for a contact center enterprise. It utilizes a VoiceXML V2.0 compliant browser for application programming interfaces.

IBM WebSphere Voice Server for Multiplatforms V5.1.2/V5.1.3 and the Cisco Customer Voice Portal V3.1 support full open standards using the:

- ▶ Speech Recognition Grammar Specification (SRGS) V1.0 for grammars
- ▶ Semantic Interpretation for Speech Recognition (SISR) for semantic interpretation
- ▶ Speech Synthesis Markup Language (SSML) V1.0 for Text To Speech markup
- ▶ Voice eXtended Markup Language (VoiceXML) V2.0 for speech application markup
- ▶ Media Resource Control Protocol (MRCP) V1 Draft 4 for interoperability between the Interactive Voice Response (IVR) and speech server.

This IBM Redpaper is a guide for integrating IBM Websphere Voice Server for Multiplatforms V5.1.2/V5.1.3 and Cisco Customer Voice Portal V3.1.

In this paper, we consider best practices as applied to Voice User Interface (VUI) design. We implement and deploy a simple voice-enabled application using Cisco Customer Voice Portal and WebSphere Voice Server V5.1.2/V5.1.3.

We demonstrate how to develop, test, and deploy a simple voice-enabled application using the VoiceXML markup language. We also demonstrate the use of Automatic Speech Recognition (ASR) and Text to Speech (TTS) voice technologies through examples.

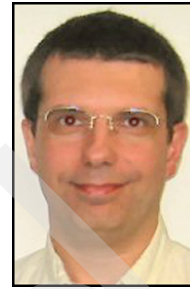
With this paper, you can tailor and configure WebSphere Voice Server and Cisco Customer Voice Portal in basic and advanced topologies.

We assume you have a basic knowledge of Interactive Voice Response (IVR) systems and voice-enablement of applications using VoiceXML, Cisco Customer Voice Portal, WebSphere Voice Server, and WebSphere Application Server.

## The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

**James Chamberlain** is a Senior Software Engineer and certified Senior IT Specialist. He is a project leader at the ITSO, Raleigh Center. He has over 24 years experience in the IT industry and specializes in pervasive computing technologies. His areas of expertise include e-commerce, pervasive computing, portals, AIX®, Linux®, and Java programming. He also architects, designs, and develops solutions using J2EE, XML, Web Services, and IBM software products including WebSphere and DB2®. Before joining the ITSO, James worked for IBM Global Services on e-commerce system development for IBM Business Partners. He majored in Computer Science at Iowa State University.



**Guy Kempny** is an IT software specialist in IBM Australia. He has 17 years of experience in the IT industry. His expertise covers a broad range of areas that include support for Intel® software, OS/390®, speech software. He is an IT specialist for WebSphere and Lotus® software. Previous experiences include project managing IT solutions for IBM's corporate sports sponsorship and large-scale events. Worth mentioning was the role of Program Manager for the IBM Olympic Project Office for the Sydney 2000 Olympic Games. Before joining the IBM Marketing team as a Technical Marketing Specialist, he was part of the WebSphere software technical team. Guy has participated in several Redbooks™ on speech and served as the ITSO Resident Team Leader for this residency.



**Jeff Wolfeld** is a Technical Leader and lead architect for Cisco's Customer Voice Portal product. He joined Cisco in 2000 to concentrate on IP contact center and related products. Mr. Wolfeld has been involved in voice-related products and applications since the early 1980s, designing speech recognition products at Intel, voice mail systems at ROLM, and IVR products at Aspect Communications. At Aspect, he also served as an IVR application consultant.



**Peter Taylor** is a graduate of the Herzing Institute and is a specialist in Voice Application architecture and development. He joined SOFTEL Communications early in 2004 to focus on Voice/IVR application development and systems integration activities.





**Bill Barrus** is a Senior Software Engineer in IBM Software Group's Business Partner Technical Enablement organization. He began his IBM career doing mechanical design trade-off studies for the US Navy F-14 avionics upgrade program, moved into Computer Aided Design software and most recently contributed to IBM's Emerging Business Opportunity (EBO) challenges such as CATIA engineering analysis, IBM Workplace™ Client Technology, IBM WebSphere Everyplace® Access, and IBM xSeries® partner coordination. He is currently involved with a number of contact center partners who are integrating their products with IBM WebSphere Voice Server.



**Vance Vagell** is a VoiceXML certified Software Support Representative working at Audium Corporation in New York City. Prior to joining Audium Corporation's support team, he was a software developer who programmed for many industries including game development, investment, and education. He has over twelve years of programming experience, in languages which include C, C++, Java, VoiceXML, Lingo, Pascal, Delphi, and others. He is a graduate of Western Connecticut State University, with a BA in Computer Science.



Thanks to the following people for their contributions to this project:

**Jason Tepper**  
Audium Corporation

And a special thanks to our ITSO support staff at the International Technical Support Organization, Raleigh Center:

Margaret Ticknor  
Jeanne Tucker  
Tamikia Barrow  
Linda Robinson

Thanks to our ITSO management:

John Byrd  
Jere Cline

And a special thanks to our IBM Pervasive Computing sponsor:

Mary Fisher

## Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this Redpaper or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an email to:

[redbook@us.ibm.com](mailto:redbook@us.ibm.com)

- Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HZ8 Building 662  
P.O. Box 12195  
Research Triangle Park, NC 27709-2195

# Introduction

This chapter provides an introduction to the interoperability of WebSphere Voice Server for Multiplatforms V5.1.3 and Cisco Customer Voice Portal V3.1. These products interoperate based upon the Media Control Resource Protocol (MRCP) Version 1 draft 4 specification. This is the speech industry's standard interface for any speech client such as Voice eXtended Markup Language (VoiceXML) browser, in any Interactive Voice Response (IVR) to communicate with a speech server. This chapter contains the following:

- ▶ Overview of WebSphere Voice Server for Multiplatforms V5.1.3
- ▶ Overview of Cisco Customer Voice Portal V3.1
- ▶ Typical interactive voice response scenario
- ▶ High level architecture design
- ▶ Interoperability
- ▶ Discussion of what comes next

## 1.1 Overview of WebSphere Voice Server for Multiplatforms V5.1.x

WebSphere Voice Server V5.1.3 runs as an Enterprise Application in WebSphere Application Server V5.1.1. This extends the WebSphere Application Server benefits of reliability, scalability, and availability to WebSphere Voice Server V5.1.3. These are described in *WebSphere Voice Server for Multiplatforms V5.1.1/V5.1.2 Handbook*, SG24-6447.

In addition, WebSphere Voice Server V5.1.3 now supports the industry open standard called *Media Resource Control Protocol* (MRCP) V1 Draft 4. MRCP is described in 1.5, “Interoperability” on page 9.

### 1.1.1 WebSphere Voice Server V5.1.3 features

At the time of this writing, the latest release of WebSphere Voice Server is V5.1.3 for Linux, Microsoft® Windows® Server 2003, and AIX. WebSphere Voice Server V5.1.3 for Linux is part of the larger IBM WebSphere family of products. It can be integrated with other software and hardware telephony products to speech-enable them and their associated applications. The list of features include:

Support for the following platforms:

- ▶ Microsoft Windows Server 2003 with no Service Pack installed
- ▶ Red Hat Enterprise Linux WS/ES/AS for Intel V3.0 Update 1 or Update 3 or Update 4 (2.4 Kernel)
- ▶ SUSE LINUX Enterprise Server V8, powered by United Linux V1.0 (SLES-8) (Intel) with Service Pack 2a or 3
- ▶ AIX 5L™ V5.3, WebSphere Voice Server V4.2 for AIX

For a list of the current platforms supported, visit the following URL:

[http://www.ibm.com/software/pervasive/voice\\_server/system\\_requirements](http://www.ibm.com/software/pervasive/voice_server/system_requirements)

To get the latest WebSphere Voice Server fix packs for your operating system, please visit the WebSphere Voice Server Support Page at:

[http://www.ibm.com/software/pervasive/voice\\_server/support](http://www.ibm.com/software/pervasive/voice_server/support)

- ▶ Connectivity to multiple supported IVR vendors that support MRCP V1 Draft 4. For a list of compatible IVRs, please visit the following URL:

[http://www.ibm.com/software/pervasive/voice\\_server/ivrgateway.html](http://www.ibm.com/software/pervasive/voice_server/ivrgateway.html)

- ▶ Support for the following:

- VoiceXML V2.0 Built-in Grammar Types for all languages
- Speech Recognition Grammar Specification (SRGS) V1.0

For more information about SRGS, visit the following URL:

<http://www.w3.org/TR/2004/REC-speech-grammar-20040316>

- Speech Synthesis Markup Language (SSML) V1.0

For more information about SSML, visit the following URL:

<http://www.w3.org/TR/2004/REC-speech-synthesis-20040907>

- Semantic Interpretation for Speech Recognition (SISR) - W3C Working Draft 1, dated April 2003

For more information about SISR, please visit the following URL:

<http://www.w3.org/TR/2003/WD-semantic-interpretation-20030401>

- ▶ Grammar-based speech recognition that includes support for inline, built-in grammar, and external grammars referenced with file URI or HTTP URI
- ▶ Capability to barge-in, which allows a user to interrupt the dialog and respond to a prompt
- ▶ WebSphere Voice Server is built upon standards including leveraging the power of WebSphere Application Server through J2EE and Enterprise JavaBeans™
- ▶ WebSphere Voice Server leverages the WebSphere Application Server infrastructure for failover, recovery, scalability, and quality
- ▶ WebSphere Voice Server V5.1.3 runs as an Enterprise Application in WebSphere Application Server V5.1.1
- ▶ At-a-Glance System Administration facilitated by WebSphere Voice Server system administration utilizing the WebSphere Application Server system Administrative Console for configuration, administration, troubleshooting, and reviewing log and trace information
- ▶ WebSphere Voice Server V5.1.3 additional voice-specific administration panels to facilitate configuration, monitoring, and troubleshooting
- ▶ WebSphere Edge Server Load Balancer component
 

This allows true load balancing in a round-robin fashion and provides a mean to gracefully take machines out of service for maintenance. It provides automatic failover where the load balancer will automatically remove a non-responding server from service. It also allows you to create and manage a WebSphere Voice Server farm where you can seamlessly add or remove WebSphere Voice Server machines without reconfiguring your IVR.
- ▶ WebSphere Application Server Network Deployment
 

This allows a single WebSphere Application Server system administrator's console to manage a cell or grouping of WebSphere Voice Server V5.1.3 machines.

**Note:** For a more detailed description of the IBM WebSphere Voice Server V5.x, refer to *WebSphere Voice Server for Multiplatforms V5.1.1/V5.1.2 Handbook*, SG24-6447.

## Language support

New to WebSphere Voice Server for Multiplatforms V5.1.3 is the ability to support more than one language per server. WebSphere Voice Server for Multiplatforms V5.1.3 supports the following languages on Microsoft Windows Server 2003 Enterprise and Standard editions, IBM AIX 5L V5.3 (including LPAR), Red Hat Enterprise Linux WS/ES/AS for Intel V3.0, and SUSE SLES V8.0:

- ▶ United State English
- ▶ United Kingdom English
- ▶ Australian English
- ▶ Canadian French
- ▶ German
- ▶ Latin American Spanish
- ▶ Japanese
- ▶ Simplified Chinese

For a list of features and languages supported by WebSphere Voice Server for Multiplatforms V5.1.3 please visit the following URL:

[http://www.ibm.com/software/pervasive/voice\\_server/about](http://www.ibm.com/software/pervasive/voice_server/about)

## 1.2 Overview of Cisco Customer Voice Portal V3.1

Cisco Customer Voice Portal (CVP) V3.1 is the latest IVR system release from Cisco to deliver a sophisticated platform for the development of advanced customer self-service solutions.

Cisco CVP V3.1 uses open standards such as VoiceXML, SRGS, and SSML, to deliver flexible, standards based, next-generation voice processing. These standards offer the flexibility to interface with standard application servers or speech server such as WebSphere Application Server and WebSphere Voice Server.

To integrate with WebSphere Voice Server, Cisco CVP V3.1 uses the Media Resource Control Protocol (MRCP) for Automatic Speech Recognition (ASR) and Text to Speech (TTS). Please refer Section 1.5, “Interoperability” on page 9 for more information.

The following sections describe the Cisco Customer Voice Portal (CVP) Version V3.1 solution.

### 1.2.1 Core capabilities

*Cisco CVP V3.1* is a contact center enablement product. It provides three distinct capabilities for telephone-based contact centers. As a self-service IVR platform, it provides sophisticated IVR application development and execution capabilities. Applications are designed using an intuitive Eclipse-based Graphical User Interface (GUI), and deployed to standard Java 2 Enterprise Edition (J2EE)-based application servers. From there they execute using industry-standard VoiceXML and MRCP protocols, and can access a wide range of back-end services using either standard or custom interfaces.

When Cisco CVP V3.1 is used for its self-service IVR capabilities, it may be deployed in either a standalone configuration or in an Intelligent Call Management (ICM) configuration. In the ICM configuration, it brings two other core capabilities:

- ▶ Call queuing
- ▶ Voice over Internet Protocol (VoIP) call directing

As a queuing platform, it provides basic IVR capabilities for playing music and announcements to callers on hold. As a VoIP call director, it enables calls to transfer around the H.323 network among the IVR platform, the queuing platform, and the Automatic Call Distribution (ACD). In the case of the Internet Protocol Contact Center (IPCC), it works with the Cisco Call Manager to facilitate voice connections directly to agent IP phones.

Additionally, CVP V3.1 can be deployed in a geographically disbursed fashion known as Branch Office Model. This is a flavor of the ICM Integrated configuration which situates appliance-like equipment at potentially many thousands of branch office locations around the world, with all the call control managed from centralized data centers. Callers can dial local phone numbers to reach either self service applications or contact center agents, and the branch-based equipment serves as both the IVR and the queuing platform. Companies which deploy in this way save substantially on Public Switched Telephone Network (PSTN) toll charges as well as on network bandwidth for callers who are waiting on hold.

### 1.2.2 Highly flexible architecture

The Cisco Customer Voice Portal solution is actually a suite of products and components that are interconnected to form a highly flexible architecture. The architecture provides for extreme scalability and reliability, and allows for integration with a number of third party

components. This particular Redpaper in fact focuses on integration with two of those components:

- ▶ IBM WebSphere Application Server
- ▶ IBM WebSphere Voice Server.

### **High availability and extreme scalability**

Every component in the solution can be paired or farmed, allowing for scalability from the small enterprise level all the way to large multinational enterprises and service providers. The solution is viable and cost effective for volumes ranging from 100 to 20,000 or more simultaneous calls, and anywhere from one to several thousand locations.

All this scalability comes with a high degree of reliability. The architecture provides for N+1 through N+N redundancy, and full load balancing, and failover capabilities across components and geographic locations. The system automatically routes new calls around failed components, and even provides for configurable safety-net handling of existing calls in the event of a call-affecting failure.

### **Third-party integrations**

Because every customer has preferences, and many customers already have substantial investments in existing contact center equipment, the Cisco Customer Voice Portal solution is designed to integrate with a number of third party products.

For speech recognition and text to speech services, Cisco CVP integrates using the industry-standard MRCP protocol to ASR and TTS servers offered by Nuance, Scansoft, and IBM (we used the IBM WebSphere Voice Server in this Redpaper). For contact center agent management, desktop applications, and telephone control, CVP integrates with Cisco's IP Contact Center (IPCC) product, leading to a fully IP-based deployment. It also integrates with Time Division Multiplexing (TDM)-based ACD systems from vendors such as Avaya, Nortel, Aspect, Rockwell, and many others. And these are not lightweight integrations. In most cases, the solution can track the states and statistics associated with individual agents and is able to treat a network of interconnected ACDs from multiple vendors as a single world-wide virtual call center.

Cisco CVP provides sophisticated IVR functionality as one of its core capabilities, but many customers already have significant investments in older TDM-based IVR systems. Most major IVR vendors such as Periphonics, Intervoice/Brite, and Aspect have built-in support for Cisco's open ICM/Voice Response Unit (VRU) Interface Protocol. Cisco CVP invoke self service applications on any IVR system which is integrated in this way.

Finally, the product supports telephony and signaling interfaces with most major PSTN carriers world-wide. Network pre-routing dips are supported using all major carrier network protocols, as is a broad range of T1/T3, E1/E3, and analog trunking protocols. Calls may be transferred among IVRs, ACDs, and queuing platforms using a variety of methods including H.323 signaling, Dual Tone Multi-Frequency (DTMF)-outpulse take-back-and-transfer, and 2-B Channel Transfer.

## **1.2.3 The Cisco CVP solution**

This section lists and describes at a very high level each of the components in a Cisco Customer Voice Portal solution deployment. We have separated the list of components into two subsections:

- ▶ Those which are required for the Cisco CVP Standalone configuration
- ▶ Those that would be added in order to deploy Cisco CVP in its ICM Integrated configuration

## **Standalone IVR**

A Cisco CVP Standalone IVR deployment consists of all the equipment necessary to develop and run sophisticated VoiceXML-based IVR applications. These applications can interface with back-end databases, Web services, and so on, and generally conduct transactions with the corporate data environment. However, only minimal call control capabilities are supported in this deployment. In particular, there is no concept of an agent, a skill group, an ACD, or queuing.

### ***Cisco Gateway***

A Cisco CVP Standalone IVR solution may use any of a number of Cisco voice gateway products. The gateway provides two primary functions:

- ▶ It terminates TDM connections (T1, E1, and so on).
- ▶ It executes the VoiceXML Browser.

In the process of executing VoiceXML pages, it must fetch them from the Cisco CVP VoiceXML Server, render them using audio files fetched from a Media Server, TTS generated by an MRCP-based TTS Server, and accept human speech input using an MRCP-based ASR Server.

### ***Cisco CVP VoiceXML Server***

The Cisco CVP VoiceXML Server is a web application running under either Tomcat or IBM WebSphere Application Server V5.1.1. It serves up VoiceXML pages as requested by the Gateway, and accepts their results. It also uses a standard J2SE™ or J2EE environment to interact with other back-end servers and services in the enterprise. Applications running on this device are developed using the CVP VoiceXML Studio, and deployed here for execution.

### ***Cisco CVP VoiceXML Studio***

The Cisco CVP VoiceXML Studio is a state-of-the-art graphical environment for designing and creating VoiceXML Server applications. The graphical framework is based on Eclipse, an open source development environment which has become the defacto standard for Java-based application development of all kinds.

With Cisco CVP VoiceXML Studio, the application designer can mix and match predefined elements from a palette, extend palette elements with additional Java code, and define new elements which can then be placed right on the palette for use by multiple applications. Off-the-shelf plug-ins are also available which can be used to extend and enhance the VoiceXML Server's capabilities.

### ***MRCP Server for ASR and TTS***

Media Resource Control Protocol (MRCP) is an industry-standard protocol for controlling media operations in general, but it is used in this environment for its ability to control ASR and TTS servers. In this Redpaper we focus on IBM WebSphere Voice Server, but Nuance and Scansoft MRCP servers are also supported within the Cisco CVP environment.

WebSphere Voice Server is described in detail beginning in Section 1.7, "WebSphere Voice Server for Multiplatforms V5.1.3" on page 10.

### ***Media Server***

Despite the similarity of names, the Media Server is not related to MRCP. It is a simple web server from any vendor, running on any appropriate hardware that can serve media files (such as WAV) on request. The Gateway sends an HTTP GET request to the media server, and the Media Server simply returns the file.

In the examples in this Redpaper, we use the VoiceXML Server itself as the Media Server.



### ***Cisco Content Services Switch***

The Cisco Content Service Switch (CSS) is an optional component, which when present, substantially enhances the CVP high availability design. It adds load balancing and failover capabilities to the Gateway's Voice Browser, enabling automatic redundancy and load balancing across CVP VoiceXML Servers, MRCP Servers, Media Servers, and (in ICM integrated implementations) CVP Call Control Servers. Without a CSS, a CVP solution can implement very basic failover capabilities across these devices, but not load balancing.

### **Integrated with ICM**

An ICM Integrated deployment includes all components described in Section 3.2, "ICM Integrated Deployment" on page 49, plus those which enable more sophisticated H.323 call control and interaction with ICM.

### ***VoiceXML Gateway***

In the standalone configuration, only one gateway is used (though it may be deployed in a farm for scalability and redundancy purposes). That gateway performs both ingress and VoiceXML functions. In the ICM Integrated configuration as well, many customers do combine both ingress and VoiceXML activities on a single (farmed) gateway. However the option also exists to split those functions onto two separate gateways, which we will call the Ingress Gateway and the VoiceXML Gateway. The decision to do so is mainly economic, and depends on the ratio of callers who are talking to agents to callers who are either in queue or performing self-service activities.

Ingress gateways, which do not need to execute VoiceXML, MRCP, and HTTP, take less CPU per call, and therefore have a higher call carrying capacity. VoiceXML gateways that do not need to convert TDM to VoIP and back require no TDM cards and no Digital Signal Processors (DSPs), and therefore cost significantly less. Cisco advises customers and partners to price out both methods and choose whichever carries the more favorable cost.

In this Redpaper, when discussing the ICM Integrated configuration, we typically discuss the Ingress and VoiceXML gateways as though they are separate physical devices. Remember, however, that the option exists to combine them into the same device.

### ***Cisco CVP Call Control Server***

This server has two functions. First, it acts as a protocol converter between the ICM/VRU Interface Protocol and H.323 for call control operations. Second, it interprets Cisco CVP's built-in IVR building block operations, known as Microapplications (Microapps) and formulates VoiceXML instructions for the VoiceXML Gateway to render. Microapps provide basic IVR capabilities such as prompt and collect, play media files, and input information using ASR. While complex IVR applications can be built using these building blocks, they are much better suited to simple prompt and collect applications, and are most commonly used for playing music and announcements on hold. Customers who do not require sophisticated self-service applications will typically use these Microapplications, and not purchase Cisco CVP VoiceXML Server.

### ***Cisco Gatekeeper***

The Gatekeeper, like the Gateways, is a Cisco Internet Operating System (IOS)-based device. It is, in fact, a Cisco gateway running a different packaging of the IOS software.

In a Cisco CVP network, the Gatekeeper is used primarily for directory lookup. It is configured with a list of dialed numbers and their corresponding IP addresses for H.323 calls. The Cisco CVP Call Control Server consults the Gatekeeper whenever it is asked to transfer a call to an agent, or to a VoiceXML Gateway. The Ingress Gateway consults it in order to find a Cisco CVP Call Control Server whenever it receives a new incoming call.

The Gatekeeper is a key component in Cisco CVP's scalability and high availability design. It manages the redundancy and load balancing across all the H.323 components and takes care to never allow calls to be sent to endpoints that are not able to handle those calls.

### ***Cisco ICM Components***

The Cisco Intelligent Contact Manager (ICM) is a multi-node product in itself and will not be discussed in detail in this Redpaper. Its purpose is to manage and monitor agents across a complex network of ACDs at multiple locations worldwide and to create the sense of a single unified *virtual contact center* for an enterprise or contact center service provider.

Cisco ICM supports ACDs from many vendors, even at the same time. But it also supports agents using IP phones connected to Cisco Call Manager. When deployed in that way, the product is known as Cisco IP Contact Center (IPCC).

When a Cisco ICM routing script determines that an appropriate agent is not available for a given call, it puts the call into queue using Cisco CVP as the queuing platform. It also uses Cisco CVP to transfer the call around the VoIP network.

### ***Cisco Call Manager***

The Cisco Call Manager is an IP-based PBX system, which manages IP phones within a workgroup, a building, or an entire enterprise. When combined with ICM, the phones can be configured as contact center agents, and they can be associated with desktop screens for Computer Telephony Interface (CTI) purposes. When ICM and Call Manager are paired in this way, the product is known as IP Contact Center, or IPCC.

Cisco CVP is an optional component to IPCC, just as Call Manager is an optional component to CVP. Typically, calls will be managed by ICM via CVP, while ICM also watches IPCC agent activity. Only when ICM decides to transfer the call to an IPCC agent, will CVP interact at all with the Call Manager.

## **1.3 Typical interactive voice response scenario**

Typically, business applications that are enabled for voice are either simple query or transaction-based applications.

For example, a customer can invoke a query application to retrieve information from the database. A simple scenario to demonstrate this is placing a call to retrieve an account balance, the local weather, movie listings, or stock quotes. This information can now be obtained from a customer contact center's IVR.

These are common examples of query applications that can be enabled for voice. Usually, the application guides the customer through a series of menus and forms aided by instructions, prompts, and choices.

The application uses either prerecorded audio files or Text to Speech (TTS), a technology that synthesizes spoken language from text. The customer can use spoken words to make selections and provide information. These words are processed and interpreted using a technology known as Automatic Speech Recognition (ASR). Once the database records are retrieved, the system presents the information by playing back audio files.

On the other hand, a transaction application is used for the purpose of performing specific tasks. Transferring funds between accounts and the buying and selling of stocks are popular examples of this type of application. In the above scenarios, the application starts by authenticating the customer through an identification process, and then uses a series of prompts and menus to guide the customer to enter the required data such as the source and

target accounts for a funds transfer transaction. The system plays instructions, prompts, and available options using prerecorded audio files or by performing Text to Speech. The customer responds using spoken commands. Once the required data is collected, the system performs the transaction and returns the result.

This functionality is achieved through the use of VoiceXML applications. For the sake of this interoperability Redpaper, we have decided to create a voice application using Cisco CVP Studio, with help from IBM Voice Toolkit for WebSphere Studio enhanced features.

For WebSphere Voice Server tools and components, please refer to *WebSphere Voice Server for Multiplatforms V5.1.1/V5.1.2 Handbook*, SG24-6447.

## 1.4 High level architecture design

Figure 1-1 illustrates a very common setup at a high level.

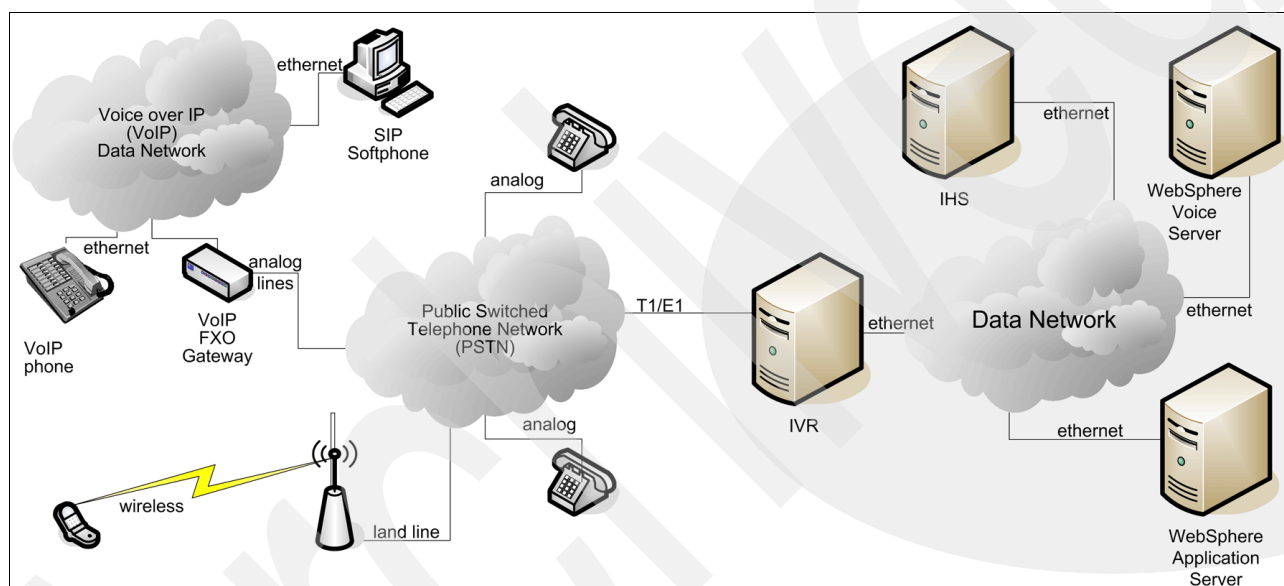


Figure 1-1 High level overview of a typical IVR to voice system setup

## 1.5 Interoperability

MRCP is defined by the Internet Engineering Task Force (IETF) and the version supported by WebSphere Voice Server is the MRCP V1 Draft 4 specification. MRCP provides the mechanism for controlling speech synthesizer and speech recognizer resources in a standardized manner. This allows any client that is MRCP compliant the ability to talk to any MRCP server.

WebSphere Voice Server V5.1.3 can interoperate with Cisco CVP V3.1 as they both support the MRCP V1 Draft 4.

For more WebSphere Voice Server interoperability information, please refer to *WebSphere Voice Server for Multiplatforms V5.1.1/V5.1.2 Handbook*, SG24-6447.

## 1.6 What comes next?

This Redpaper is a ground-up tutorial for developing and deploying Cisco Customer Voice Portal applications using IBM WebSphere Voice Server. We take a two-pronged approach to development. Having presented an overview of the products and components, we detail the process of installing each major component, performing its initial configuration, and verifying that they are all running and working together.

Given a working environment, we switch over to application development. We develop a series of four applications which progressively demonstrate the capabilities of the solution:

- ▶ Basic DTMF application to demonstrate the process of creating, deploying, and testing an application
- ▶ Basic voice-based application to show how to add ASR and TTS to the mix
- ▶ Complex voice-based application to demonstrate various capabilities of the CVP VoiceXML Server
- ▶ Complex voice application which integrates with an ICM-based contact center

The first three applications are developed using the Cisco CVP Standalone deployment, while the fourth one uses an ICM Integrated deployment.

## 1.7 WebSphere Voice Server for Multiplatforms V5.1.3

In this Redpaper, we used WebSphere Voice Server for Multiplatforms V5.1.3. There are a number of new features incorporated with this version. One of the significant differences to previous versions is that multiple languages can now be installed on one server. Below is summary list of the features of WebSphere Voice Server for Multiplatforms V5.1.3:

Supported operating systems

- ▶ AIX 5L V5.3 with Maintenance Level (ML) 3
- ▶ Red Hat Enterprise Linux WS/ES/AS for Intel V3.0 with Update 1, Update 3, or Update 4 (2.4 Kernel)
- ▶ SUSE LINUX Enterprise Server V8, powered by United Linux 1.0 (SLES-8) (Intel) with Service Pack 2a or 3
- ▶ Microsoft Windows Server 2003 Enterprise or Standard editions

**Restriction:** No Windows Service Packs are supported at this time.

- ▶ For a list of the current platforms supported, please visit the WebSphere Voice Server System requirements web page at:

[http://www.ibm.com/software/pervasive/voice\\_server/system\\_requirements](http://www.ibm.com/software/pervasive/voice_server/system_requirements)

WebSphere Voice Server for Multiplatforms V5.1.3 includes:

- ▶ IBM WebSphere Voice Server V4.2 for AIX
- ▶ Inclusion of IBM WebSphere Voice Server V4.2 for AIX, and new language support to that already provided by V5.1, V5.1.1, V5.1.2, and V4.2.
- ▶ Support for the following languages on Microsoft Windows Server 2003 Enterprise and Standard editions, IBM AIX 5L V5.3 (including LPAR), Red Hat Enterprise Linux WS/ES/AS for Intel V3.0, and SUSE SLES V8.0:

- U.S. English
  - U.K. English
  - Australian English
  - Canadian French
  - German
  - Latin American Spanish
  - Japanese
  - Simplified Chinese
- ▶ Ability to support more than one language installed on a single machine
  - ▶ Support for connectivity to multiple qualified Interactive Voice Response (IVR) vendors, including those who support the MRCP V1 draft 4.
- For a list of compatible IVRs, please visit the WebSphere Voice Server IVR/Compatibility Web page:
- [http://www.ibm.com/software/pervasive/voice\\_server/ivrgateway.html](http://www.ibm.com/software/pervasive/voice_server/ivrgateway.html)
- ▶ Support for VoiceXML V2.0 Built-in Grammar Types for all languages.
- Speech Recognition Grammar Specification (SRGS) V1.0, and Speech Synthesis Markup Language (SSML) V1.0. For more information about SRGS, visit the W3C Web site:
- <http://www.w3.org/TR/2004/REC-speech-grammar-20040316>
- ▶ Speech Synthesis Markup Language (SSML) V1.0
- For more information about SSML, please visit the W3C web site:
- <http://www.w3.org/TR/2004/REC-speech-synthesis-20040907>
- ▶ Support for Semantic Interpretation for Speech Recognition (SISR) - W3C Working Draft 1, dated April 2003
- For more information about SISR, please visit the W3C Web site:
- <http://www.w3.org/TR/2003/WD-semantic-interpretation-20030401>
- ▶ Grammar-based speech recognition that includes support for inline, built-in grammar, and external grammars referenced via file URI or HTTP URI.
  - ▶ Capability to barge-in, which allows a user to interrupt the dialog and respond to a prompt
  - ▶ WebSphere Voice Server is built upon standards including leveraging the power of WebSphere Application Server through J2EE and Enterprise JavaBeans
  - ▶ WebSphere Voice Server leverages the WebSphere Application Server infrastructure for failover, recovery, scalability, and quality
  - ▶ WebSphere Voice Server V5.1.x runs as an Enterprise Application in WebSphere Application Server V5.1.1
  - ▶ System Administration facilitated by WebSphere Voice Server system administration utilizing the WebSphere Application Server Administrative Console for configuration, administration, troubleshooting, and reviewing log and trace information
  - ▶ WebSphere Voice Server V5.1.x additional voice-specific administration panels to facilitate configuration, monitoring, and troubleshooting
  - ▶ WebSphere Edge Server Load Balancer component
- This allows true load balancing in a round-robin fashion and provides a means to gracefully take machines out of service for maintenance. It provides automatic failover where the load balancer will automatically remove a non-responding server from service. It also allows you to create and manage a WebSphere Voice Server farm where you can seamlessly add or remove WebSphere Voice Server machines without reconfiguring your IVR.
- ▶ WebSphere Application Server Network Deployment

This allows a single WebSphere Application Server system Administrative Console to manage a cell or grouping of WebSphere Voice Server V5.1.x machines.

## 1.8 WebSphere Voice Server for Multiplatforms V5.1.3 integration

The purpose of this Redpaper is to discuss how WebSphere Voice Server V5.1.3 is installed and configured within a Cisco CVP IVR solution. We created two environments:

- ▶ A basic deployment environment for development
- ▶ An advanced deployment environment for a real-life solution

### 1.8.1 Basic deployment environment for a development

In this solution, a simple environment was created to help the developer plan, build, and test an application. Minimum hardware was used to build this solution. This environment allowed the developer to test the application in several ways:

- ▶ Using a traditional phone connected through a PBX that, in turn, connected to the Cisco IVR
- ▶ By way of a soft phone application on a PC that used the H.323 protocol to communicate with the Cisco IVR

**Note:** Cisco IVR needs to be enabled for this feature. See the Appendix A, “Cisco Router Configurations” on page 347 for the Cisco 1751-V configuration file for more information.

### 1.8.2 Advanced deployment environment for production

Here, we integrated Cisco CVP V3.1 and WebSphere Voice Server for Multiplatforms V5.1.3 into a complex solution where redundancy and added functionality was included. We also included features that would be used in a real call center. For the Cisco side, we included the following functions:

- ▶ Cisco ICM components
- ▶ Cisco Call Control Server
- ▶ Cisco Gatekeeper

For the WebSphere Voice Server for Multiplatforms V5.1.3 side, we provided the following:

- ▶ Multiple Voice Servers each running several languages.
- ▶ WebSphere Edge Server Load Balancer V5.1.1.41 to manage balance the work load of incoming requests to a voice server farm.

# Basic deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3

This chapter describes how to deploy a simple IVR solution using Cisco CVP V3.1 and WebSphere Voice Server for multiplatforms V5.1.3. The following topics are discussed:

- ▶ Introduction to solution
- ▶ Overall environment
- ▶ Installation and configuration steps for Cisco CVP V3.1 components
- ▶ Installation and configuration steps for WebSphere Voice Server for Multiplatforms V5.1.3
- ▶ Cisco 1751-V Modular Access Router setup and configuration.

**Important:** This solution is useful only for a development or test environment. We do not recommend this solution for a production environment. Refer to Chapter 3, “Advanced deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3” on page 47, which describes a deployment solution more appropriate for a production environment.

## 2.1 Architecture of a basic deployment solution

This basic deployment solution is intended to be used in development or test environments. Its main restriction is the number of concurrent calls that it can handle. A basic voice solution consists of several components to answer and manage calls.

The solution consists of several main components:

- ▶ A telephony environment that has an IVR to accept, process, and direct calls
- ▶ A voice browser that runs VoiceXML applications that interact with the user on the telephone
- ▶ A voice server that offers ASR and TTS services. These two services are invoked at different times by the running VoiceXML application

And optionally, one or more of the following components:

- ▶ A Web server that hosts static VoiceXML documents, external grammars, and prerecorded audio.
- ▶ An application server that hosts applications that dynamically produce VoiceXML. The application server can link to other back-end applications and retrieve and update database information.

### 2.1.1 Cisco CVP V3.1 and WebSphere Voice Server in a basic deployment infrastructure

Creating an end-to-end solution requires building an isolated telephony infrastructure. IBM servers were used in this solution. These servers were connected on an isolated 100 Mbps, full-duplex switched network.

We used a Gordon Kapes System 930 Telephony Simulator as our Central Office (CO) or Private Branch eXchange™ (PBX) in our telephony environment. Our Gordon Kapes system was configured with two digital T1 boards, so we could connect up to two T1 trunks to one IVR or run two different IVRs at the same time.

For simple testing, four analog ports were provided to make calls. This meant four analog phones can be connected directly. There is also an amphenol connector to connect additional phone lines. In addition, the Gordon Kapes System 930 Telephony Simulator allows the simulation of 30 concurrent calls.

The basic deployment was installed on a Windows based operating system. A mixture of Microsoft Windows 2000 and Windows 2003 servers were used. In each server the supported fix packs were installed to meet supported level for that component.

### 2.1.2 Basic deployment solution topology

Figure 2-1 on page 15 depicts the basic deployment infrastructure used by the Redpaper team. If you substitute your telephony infrastructure in place of the Gordon Kapes system, then this deployment environment is capable of scaling up to the appropriate limits of your hardware. This basic deployment solution can support one to two trunks worth of speech applications. However, this deployment does not provide any failover, recoverability, maintainability, or serviceability. If one of the machines must be serviced or fails, then your environment will be unavailable to handle calls.



For the basic deployment a Cisco 1751-V Modular Access Router was used with a T1 interface. The ITSO lab used a Cisco 1751-V that had only one DSP. This meant it could only handle six concurrent calls.

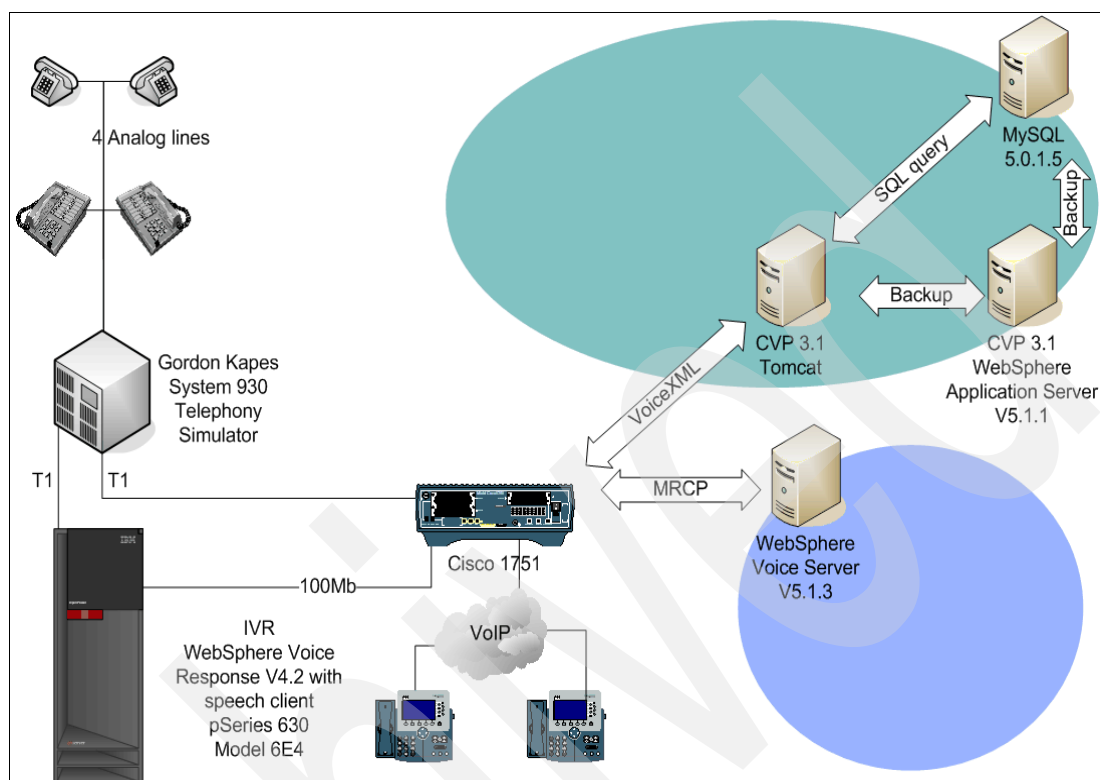


Figure 2-1 Basic deployment of Cisco CVP V3.1 and a Cisco 1751-V Modular Access Router

## 2.2 Cisco Customer Voice Portal VoiceXML server

The Cisco Customer Voice Portal comprises of several different components. Each of these components does a different function within the Cisco CVP deployment.

For the Cisco CVP VoiceXML server, we installed and tested two different application servers. Cisco CVP supports both Tomcat and IBM WebSphere Application Server V5.1.1.3. TomCat is provided with CVP V3.1.

**Restriction:** For WebSphere Application Server, an additional license is required because the WebSphere Application Server that is bundled with WebSphere Voice Server is a restricted license for running WebSphere Voice Server only. In other words, you cannot run other J2EE applications on the WebSphere Voice Server's copy of WebSphere Application Server.

For simplicity, the following steps show both the WebSphere Application Server and TomCat installation. Where they are different, the necessary steps are explained.

### 2.2.1 Software

The Cisco Customer Voice Portal CVP VoiceXML Server is supported on the following Operating System platforms:

- ▶ Microsoft Windows 2000 Server with Service Pack 4 (SP4).

**Note:** Service Pack 4 must be installed separately. Operating system CDs with SP4 embedded are not supported.

- ▶ Sun™ J2SE V1.4.2 or later.

**Note:** The J2SE V1.4.2 SDK is installed when either the Full or Custom installation option of Cisco CVP VoiceXML is selected.

The Cisco CVP VoiceXML Server requires an application server to function. At the time of this writing, only two application servers are certified:

- ▶ Tomcat V4.1.24. This is installed when either the Full or Custom installation option of Cisco CVP VoiceXML is selected.
- ▶ IBM WebSphere Application Server V5.1.1.3.

**Note:** The installation of WebSphere Application Server is only valid in the CVP VoiceXML Server or Customer installation modes.

## 2.3 Installing Cisco Customer Voice Portal VoiceXML server

The Cisco Customer Voice Portal VoiceXML server installation process is now described in detail.

### Step 1. Starting the installation

Follow these steps to begin the installation:

1. Log on to the system. This user ID must be the Administrator or a user that is a member of the Administrator group.
2. Insert the Cisco CVP CD in the CD-ROM drive.
3. Open Windows Explorer and navigate to the CD-ROM drive. Double-click the file called, **CiscoCVP.exe**.

The dialog window open as shown in Figure 2-2. It might take some time before the install process moves from this dialog.

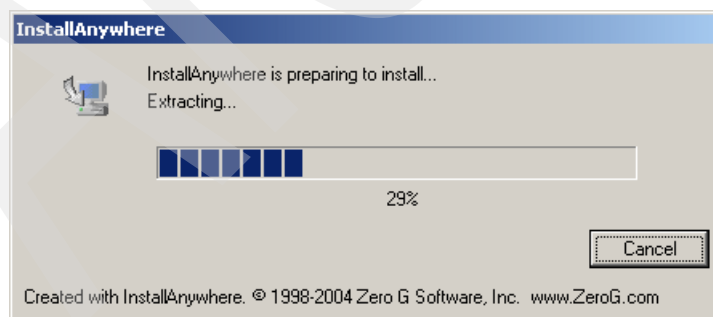


Figure 2-2 Cisco CVP install setup

The InstallAnywhere welcome dialog box opens, as shown in Figure 2-3 on page 17. Click **Next**.

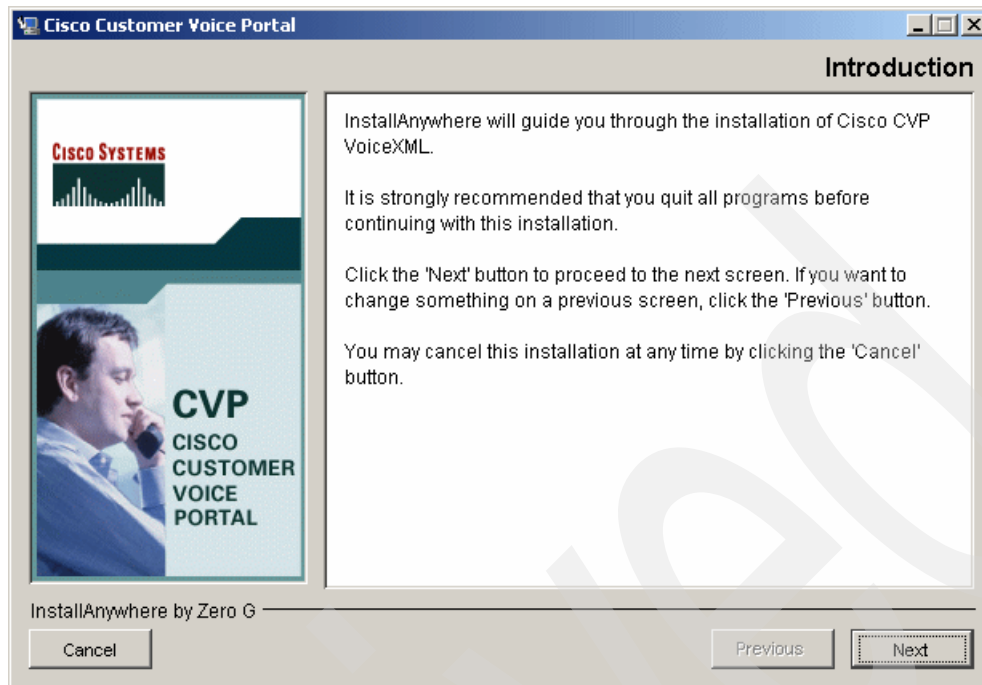


Figure 2-3 InstallAnywhere welcome page

4. The Cisco License agreement page opens, as shown in Figure 2-4. To continue the installation, you must accept the license agreement. Click **Next**.

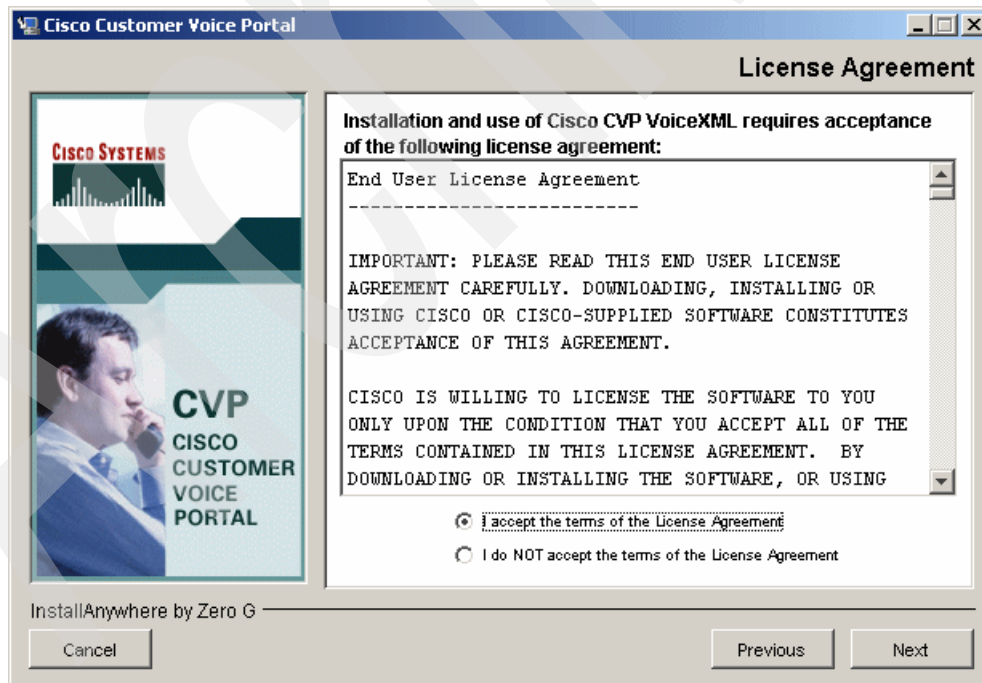


Figure 2-4 Cisco License Agreement

The VoiceXML server requires an application server. The Cisco CVP installation CD has a copy of Tomcat Application Server, however, WebSphere Application Server is also

supported. For this Redpaper both application servers were used. As each has a different installation process, both have been documented.

## Cisco CVP VoiceXML Server installation using Tomcat

These instructions continue from the License Agreement dialog box.

1. This installation is for the VoiceXML server using Tomcat, so we chose the custom configuration, as shown in Figure 2-5.

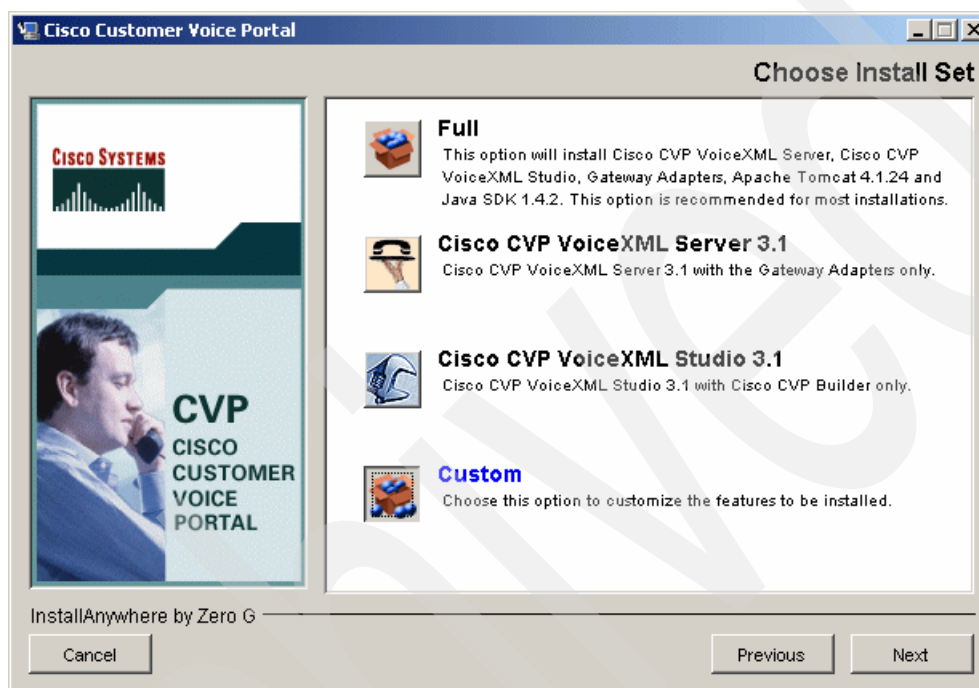


Figure 2-5 Custom install selected

2. Because this is a VoiceXML server, *deselect* **Cisco CVP VoiceXML Studio V3.1**. Click **Next**. This is shown in Figure 2-6 on page 19.

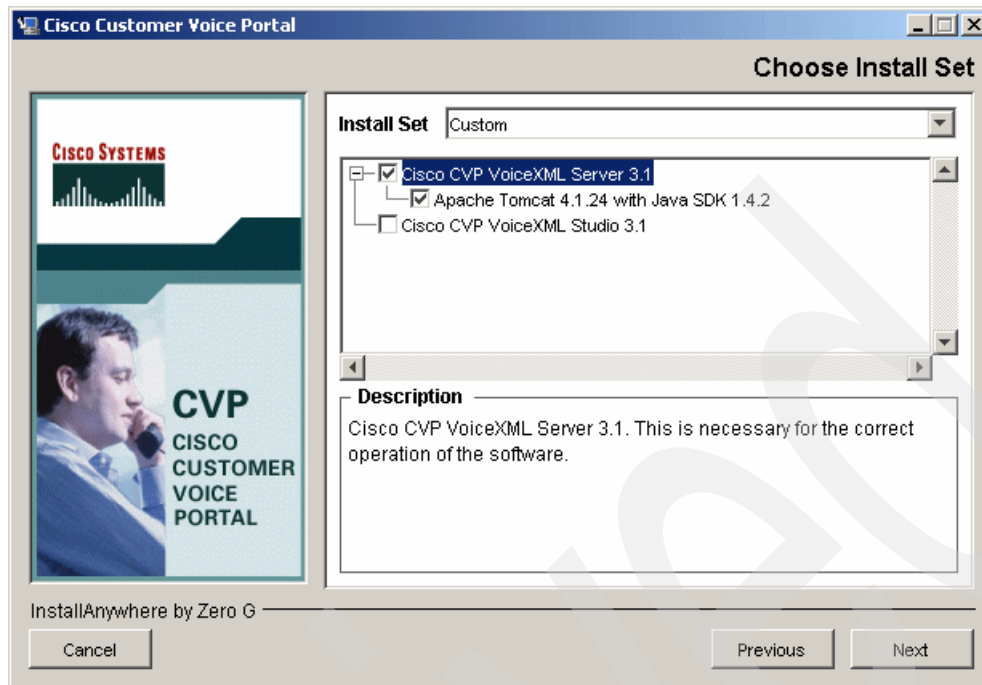


Figure 2-6 Custom options panel

3. The application server requires a user ID and password. In this case, use the system Administrator user ID and password. Leave the connector as the default. This can be seen in Figure 2-7 on page 19. Click **Next**.

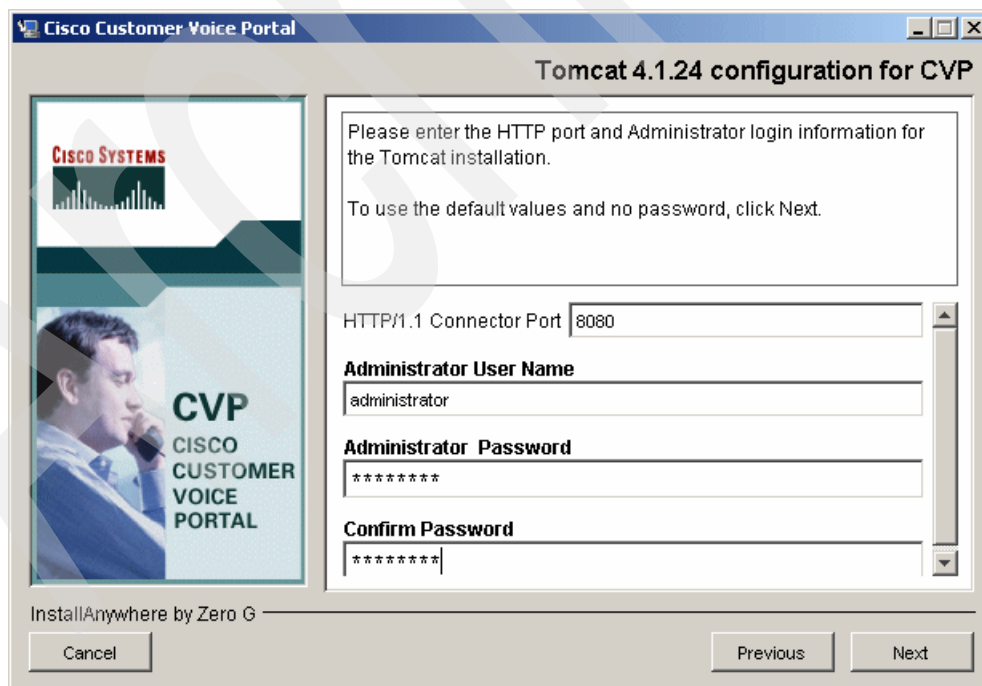


Figure 2-7 Application server login information

4. In this installation, Tomcat was made a system service. Select this as illustrated in Figure 2-8. Click **Next**.

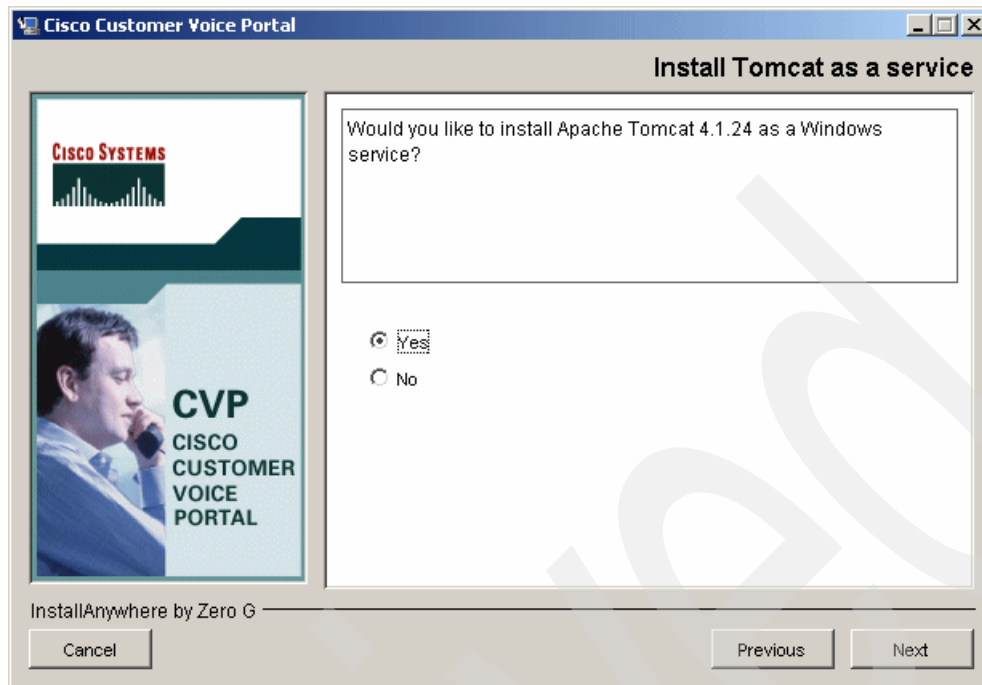


Figure 2-8 Windows service option

5. A pre-installation summary page is presented to confirm the settings used for the install. Figure 2-9 shows this dialog box. Verify that the information is correct, then click **Next** to start the actual installation.

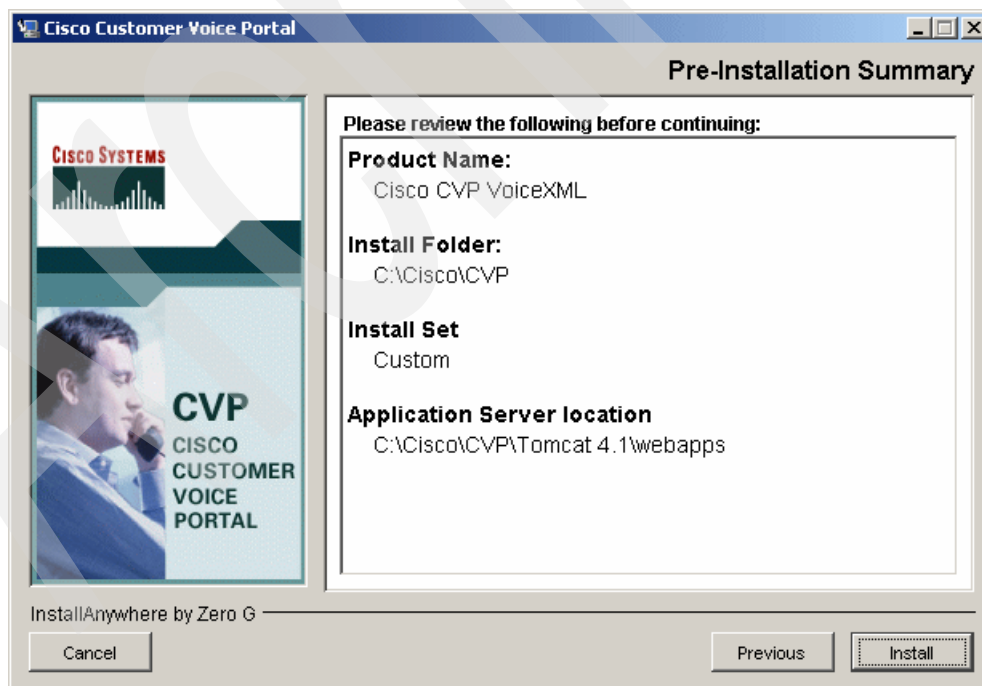


Figure 2-9 Pre-installation summary

An installation progress panel opens as shown in Figure 2-10 on page 21. The installation takes a few minutes.

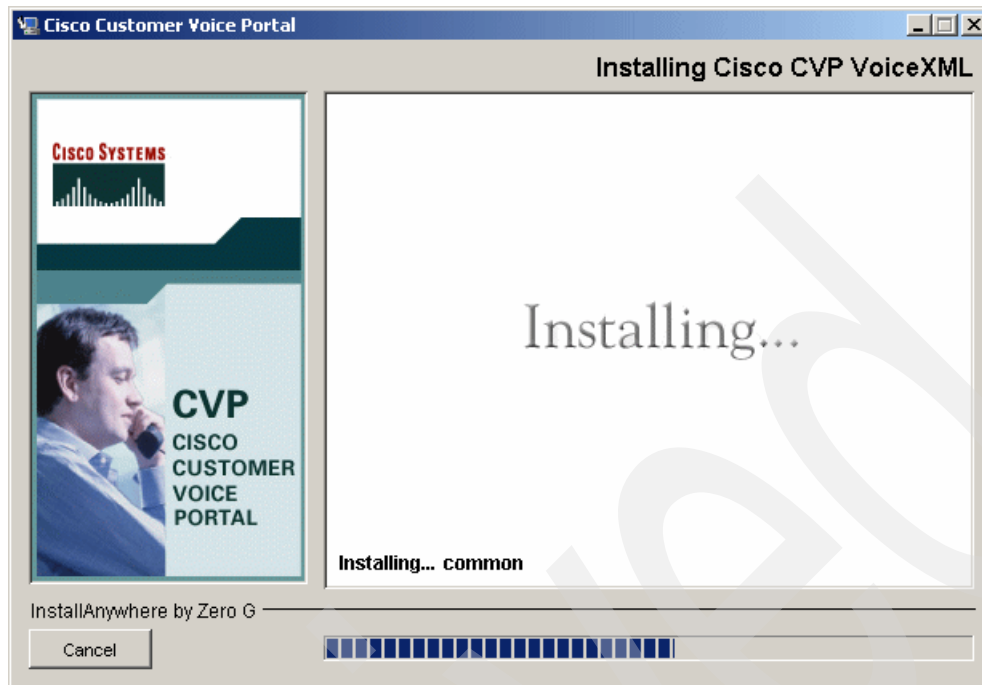


Figure 2-10 installation progress panel

6. A completion panel is presented, saying the installation was successful (see Figure 2-11). Click **Next**.

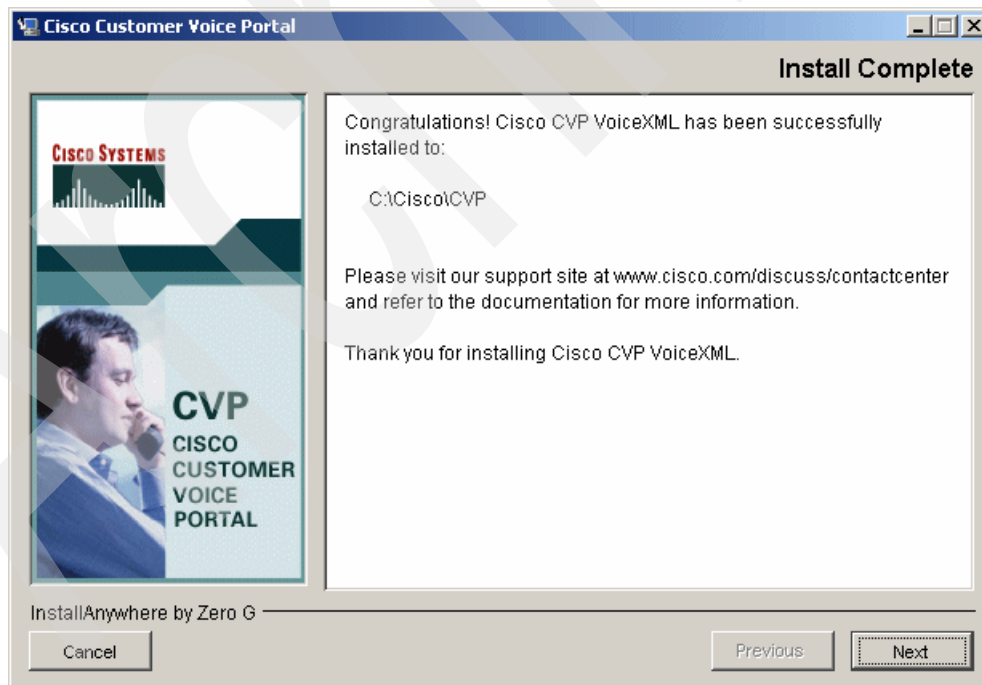


Figure 2-11 Completion panel

7. Because Cisco CVP requires the licenses to be installed before the restart, select **No** to close this dialog box.



## Cisco CVP Licensing configuration

Cisco provides a licensing file that enables the Cisco CVP VoiceXML server to run. This license is linked to a static IP address. For the purpose of this Redpaper, Cisco Systems provided these licensing files to us.

1. You must copy the License and signature files to the following subdirectory of the newly installed Cisco CVP VoiceXML server:

C:\Cisco\CVP\Server\license

Figure 2-12 shows the two files copied into the correct directory.

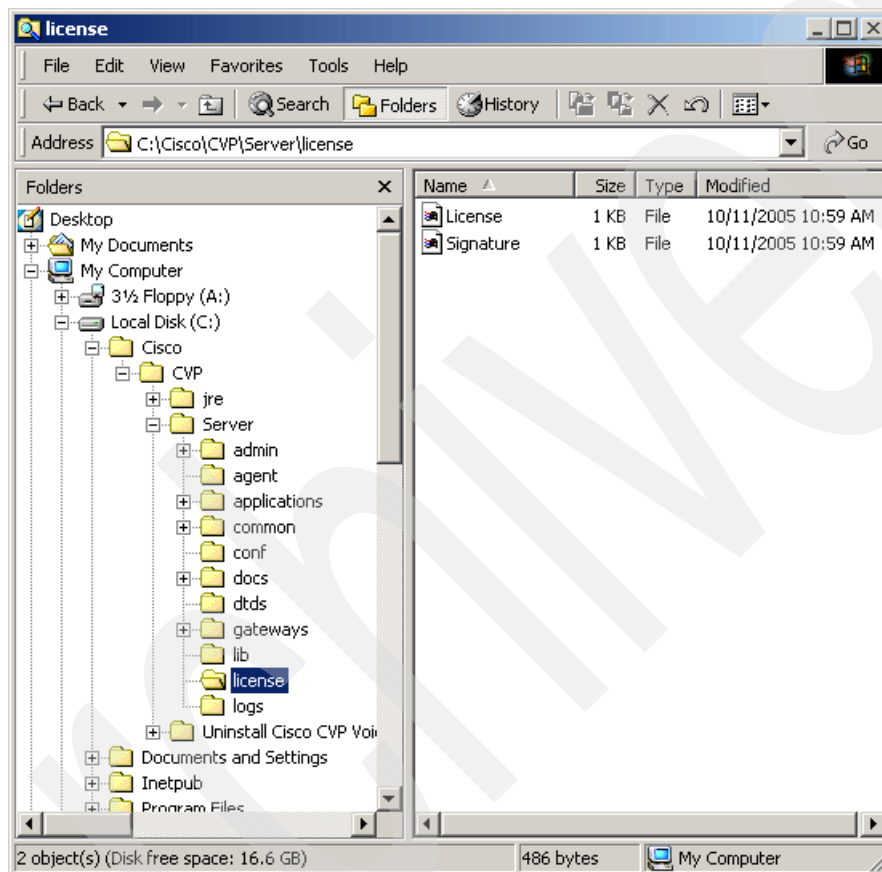


Figure 2-12 License files in position

Screenshot reprinted by permission from Microsoft Corporation

2. Reboot the server.

**Important:** The server must be rebooted at this time.

## WebSphere Application Server installation

If the Cisco Customer Voice Portal is to use IBM WebSphere Application Server, it needs to be installed first. For this installation, WebSphere Application Server V5.1.1.3 was used.

For a complete installation guide, refer to *IBM WebSphere Voice Server for Multiplatforms V5.1.1/V5.1.2 Handbook, SG24-6447*. Start the installation of WebSphere Application Server. Rather than doing a full installation, we used the following options and parameters:

1. Choose **Custom install**.



2. *Deselect* **Embedded Messaging component**.
3. Populate the Node Name and Host Name fields with the appropriate information. For this installation we used the following settings:
  - Node Name: bc1srv2
  - Host Name: bc1srv2
4. We used the default installation directory structure.
5. Ensure WebSphere Application Server and the IBM HTTP server are installed as a service.
6. Provide a user ID and password that has Administrator privileges.
7. Run the FirstSteps utility to ensure the WebSphere Application Server installed successfully.

## Cisco CVP VoiceXML Server installation using WebSphere Application Server

Follow the instructions from “Step 1. Starting the installation” on page 16 until you reach the Choose Install Set dialog box as shown in Figure 2-13 on page 23.

1. For a WebSphere Application Server installation, select **Cisco CVP VoiceXML Server V3.1**, which is highlighted in blue. Click **Next**.

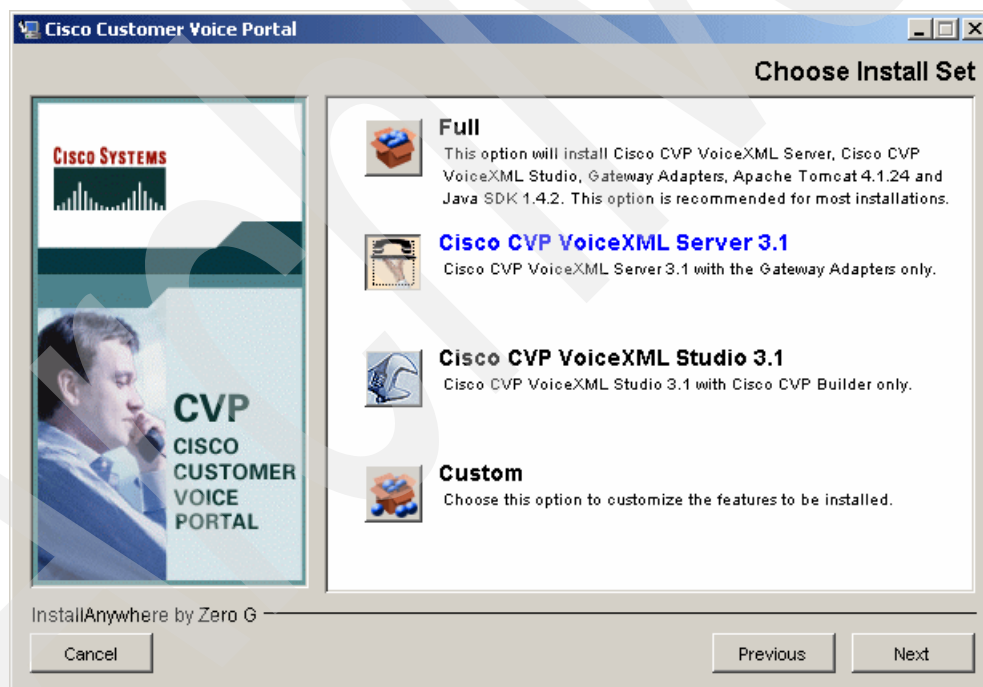


Figure 2-13 Choose Install Set

2. The default Cisco CVP directory path is shown in the Figure 2-14. We used the default. Click **Next**.

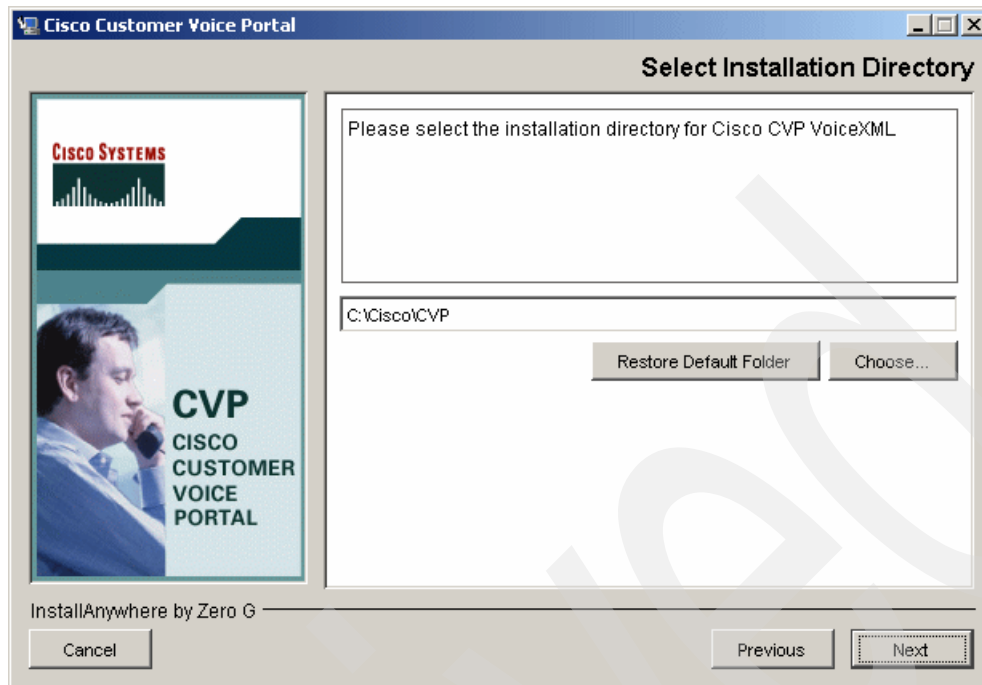


Figure 2-14 Select Installation Directory

3. For the dialog box shown in Figure 2-15, select the application server, **WebSphere 5.1**. Click **Next**.

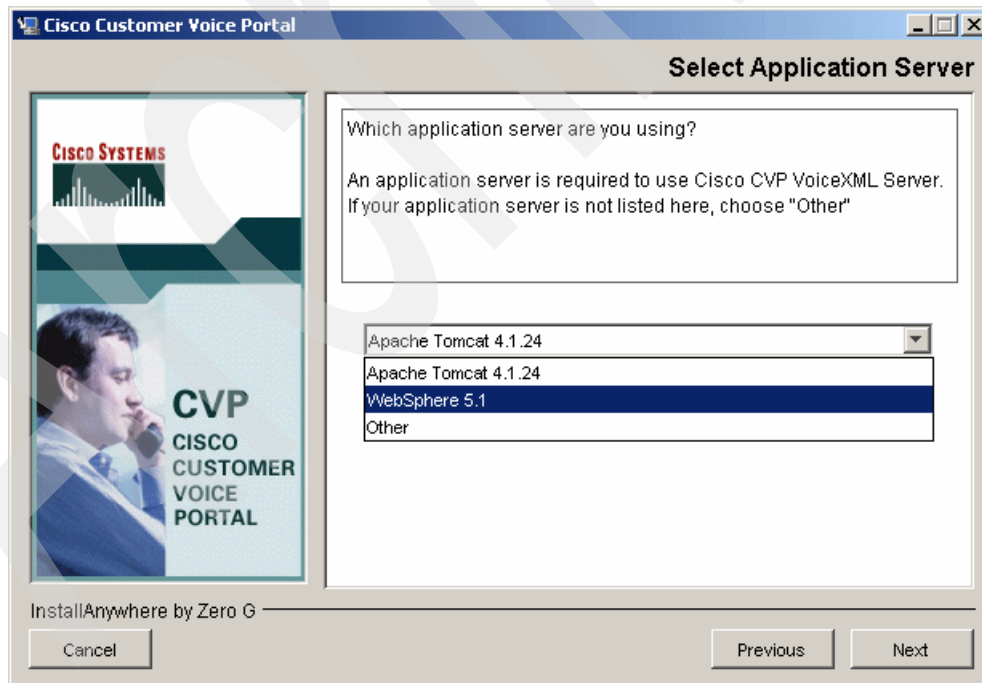


Figure 2-15 Select Application Server

4. Figure 2-16 on page 25 shows the directory structure location where the Web application will be installed. In the ITSO lab, we kept the default value. Click **Next** to continue.

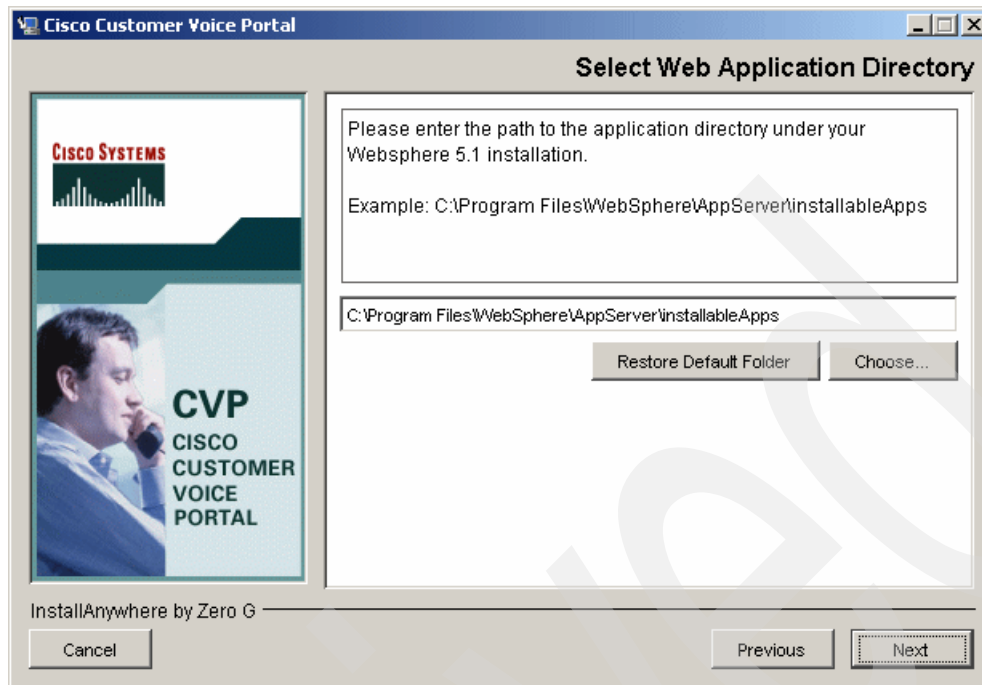


Figure 2-16 Select Web Application Directory

5. Prior to installing Cisco CVP, a preinstallation summary page is presented. This is shown in Figure 2-17. If all the parameters are correct, click **Next**.

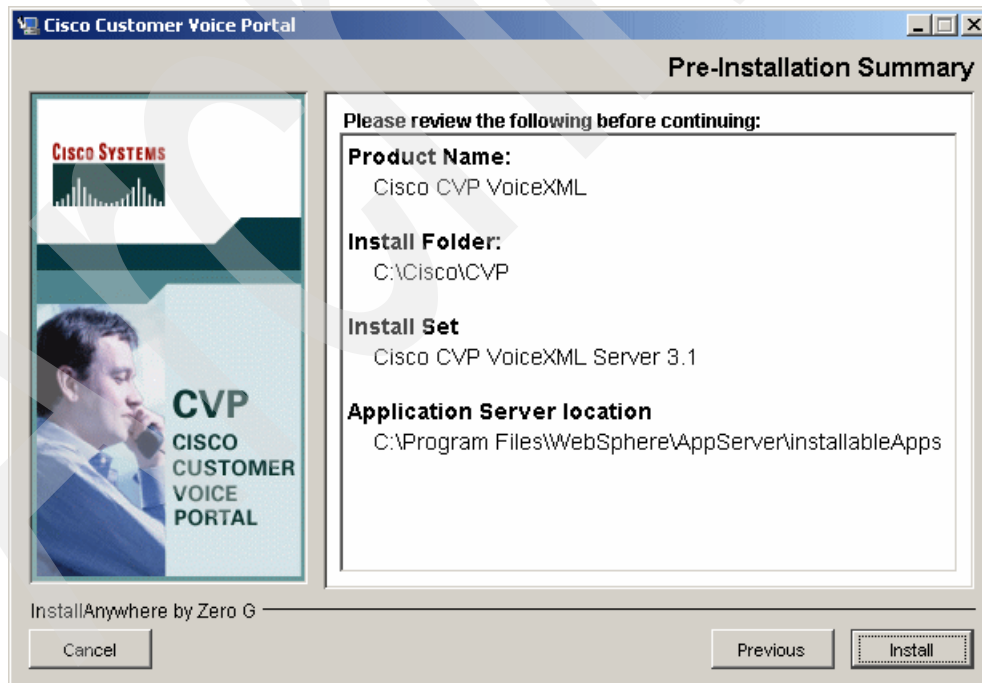


Figure 2-17 Pre-installation Summary

6. When the installation completes, a dialog box appears stating that it has finished. Click **Next**. The dialog box shown in Figure 2-18 asks if you want to restart the server. Select **No** and click **Next**.

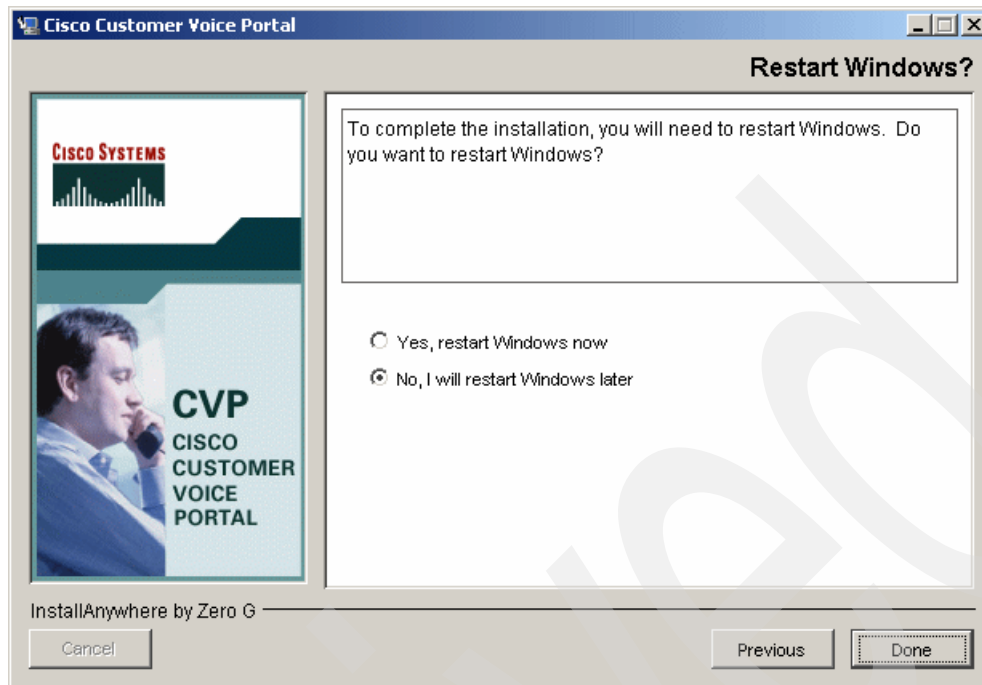


Figure 2-18 Restart Windows

### Cisco CVP Licensing configuration

Cisco provides a licensing file that enables the CVP VoiceXML server to run. This license is linked to a static IP address. For the purposes of this Redpaper, Cisco Systems provided these files to us.

1. You must copy the License and signature files to the following subdirectory of the newly installed CVP VoiceXML server:

C:\Cisco\CVP\Server\license

Figure 2-19 on page 27 shows the two files copied into the correct directory.

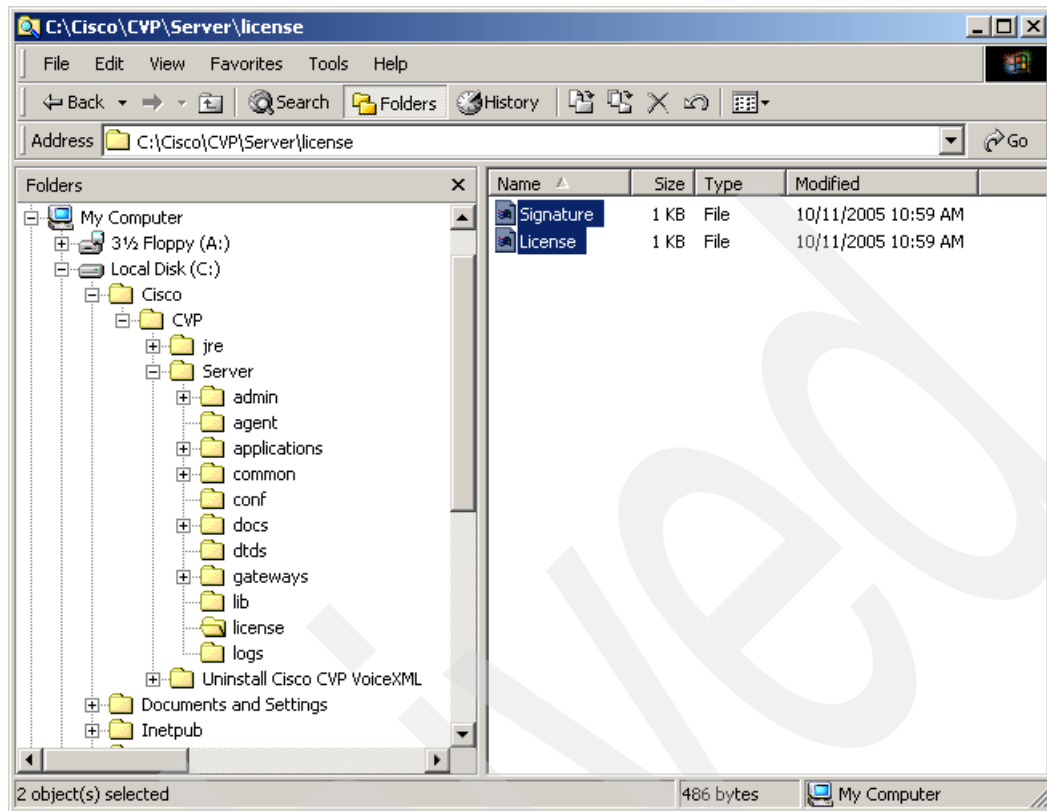


Figure 2-19 License files location

Screenshot reprinted by permission from Microsoft Corporation

2. Reboot the system.

## Configure WebSphere Application Server for Cisco CVP

This section describes the necessary configuration steps needed to for WebSphere Application Server to function with Cisco CVP. This is documented in the *Cisco CVP VoiceXML V3.1 Installation Guide*, under “Continuing CVP Server Installation on WebSphere Application Server V5.1”.

### Step 1. Start the WebSphere Application Server Administrative Console

Follow these steps to start the WebSphere Application Server Administrative Console.

1. Start a Web browser session and enter the following URL substituting in your *servername* and *userid*:

`http://servername:9090/userid`

We used the following URL for our system:

`http://bc1srv2:9090/admin`

The browser displays Figure 2-20 on page 28. Type the user Id and click **Ok**.

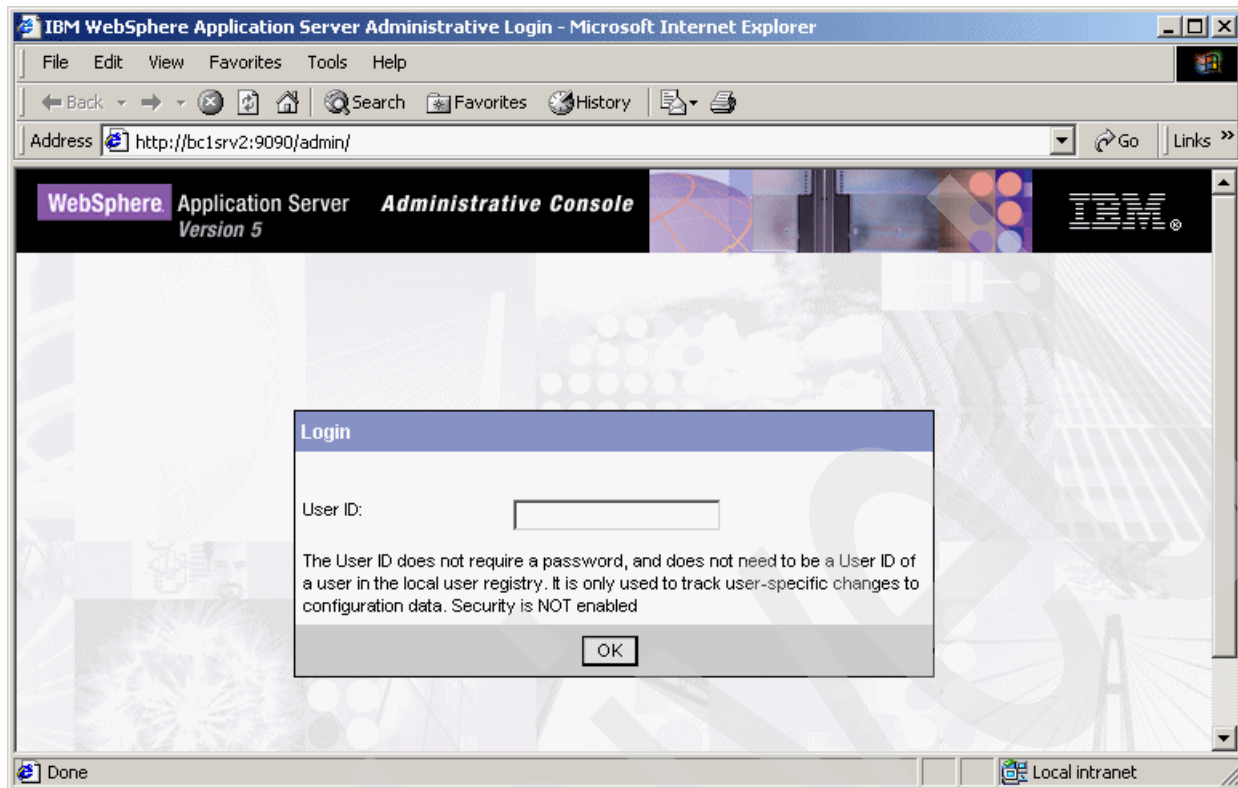


Figure 2-20 WebSphere Application Server Login screen

Screenshot reprinted by permission from Microsoft Corporation

### **Step 2. WebSphere Application Server console.**

The Administrative Console appears as shown in Figure 2-21 on page 29. The CVP Server web application needs to be installed. Navigate to:

1. Select **Applications** → **Install New Application**, then click **Install**.

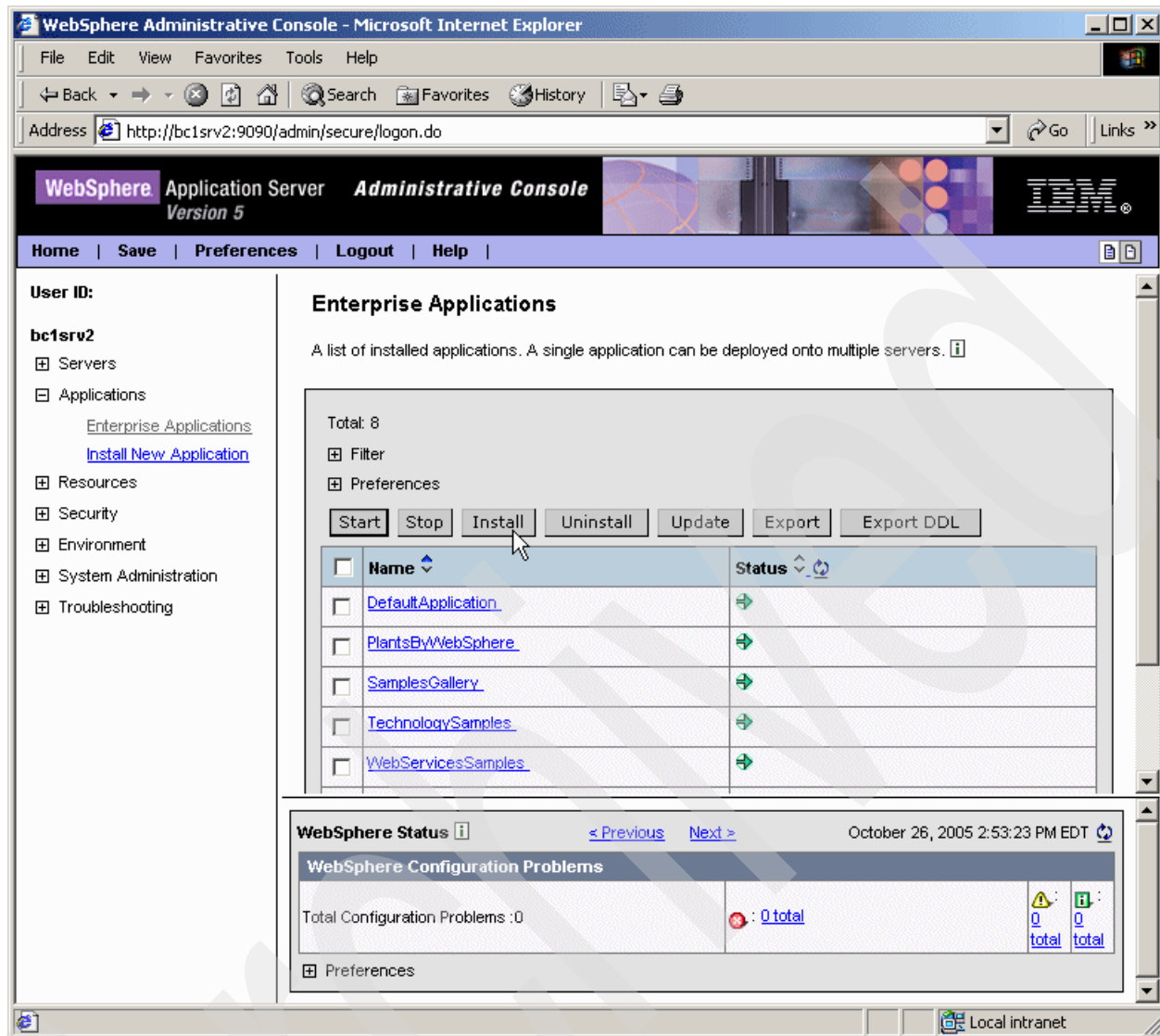


Figure 2-21 WebSphere Application Server Administration Console

Screenshot reprinted by permission from Microsoft Corporation

2. Browse the local directory to locate the CVP.war file. If the installation defaults were used for Cisco CVP, the location would be:

C:\Program files\WebSphere\AppServer\installableApps

3. After you locate the file, click **Open**.

### Step 3. Application installation

1. The Application Installation window is displayed as shown in Figure 2-22 on page 30. Enter CVP into the Context Root Field, then click **Next**.



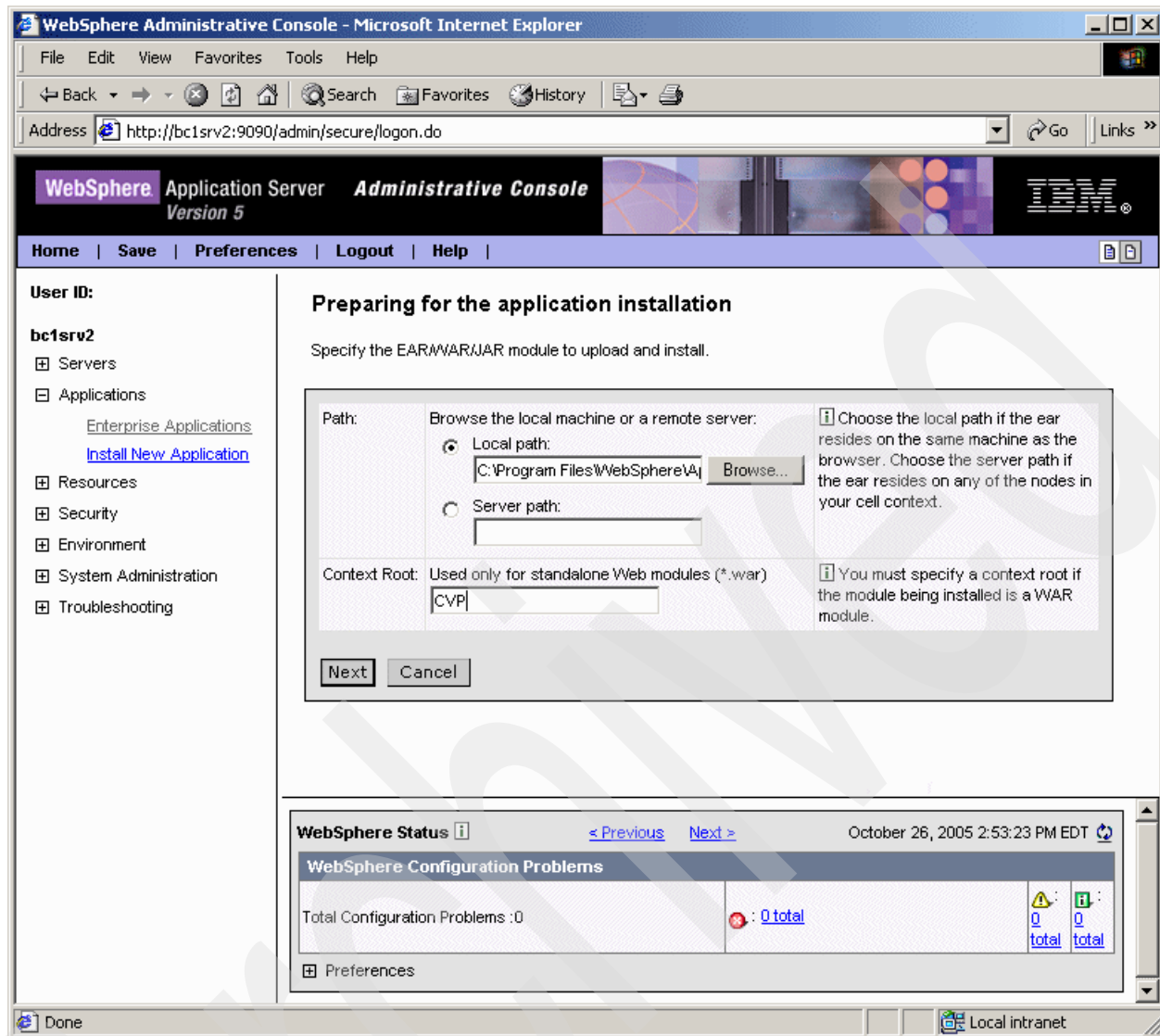


Figure 2-22 WebSphere Application Server Application installation page

Screenshot reprinted by permission from Microsoft Corporation

### Step 3. Preparing for application installation

1. The next screen as displayed in Figure 2-23 on page 31, has the bindings configuration. Leave the default values, and just click **Next**.
2. The Application Security Warnings page is displayed. Click **Next**.



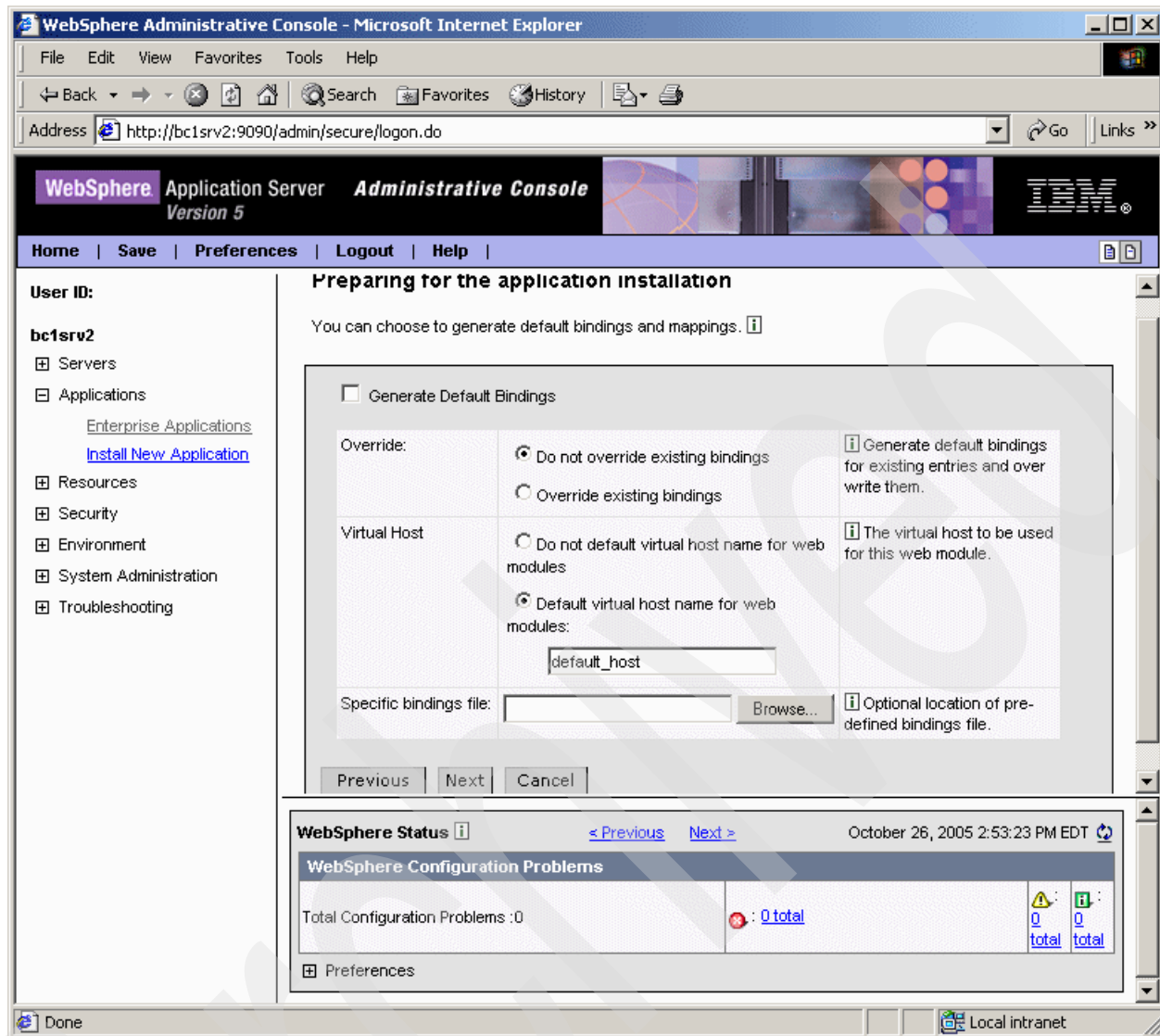


Figure 2-23 Application pre-installation configuration

Screenshot reprinted by permission from Microsoft Corporation

#### Step 4. Install New Application

The Install New Application Page has a number of configuration fields:

1. Provide options to perform the installation
  - Leave the default values and click **Next**.
2. Map virtual hosts for Web modules.
  - Select **Web Module**.
  - Select **Cisco CVP VoiceXML Server V3.1**.
  - Click **Next**.
3. Map modules to application servers:
  - Select **Module**.
  - Select **Cisco CVP VoiceXML Server V3.1**.
  - Click **Next**.
4. View the Summary:

- Leave default values.
- Click **Finish**.

### Step 5. Installation Completion

It might take a few moments before the installation results are displayed. The installation process generates messages. A message will be visible to indicate the installation completed successfully, as shown in Figure 2-24.

Click **Save to Master Configuration**.

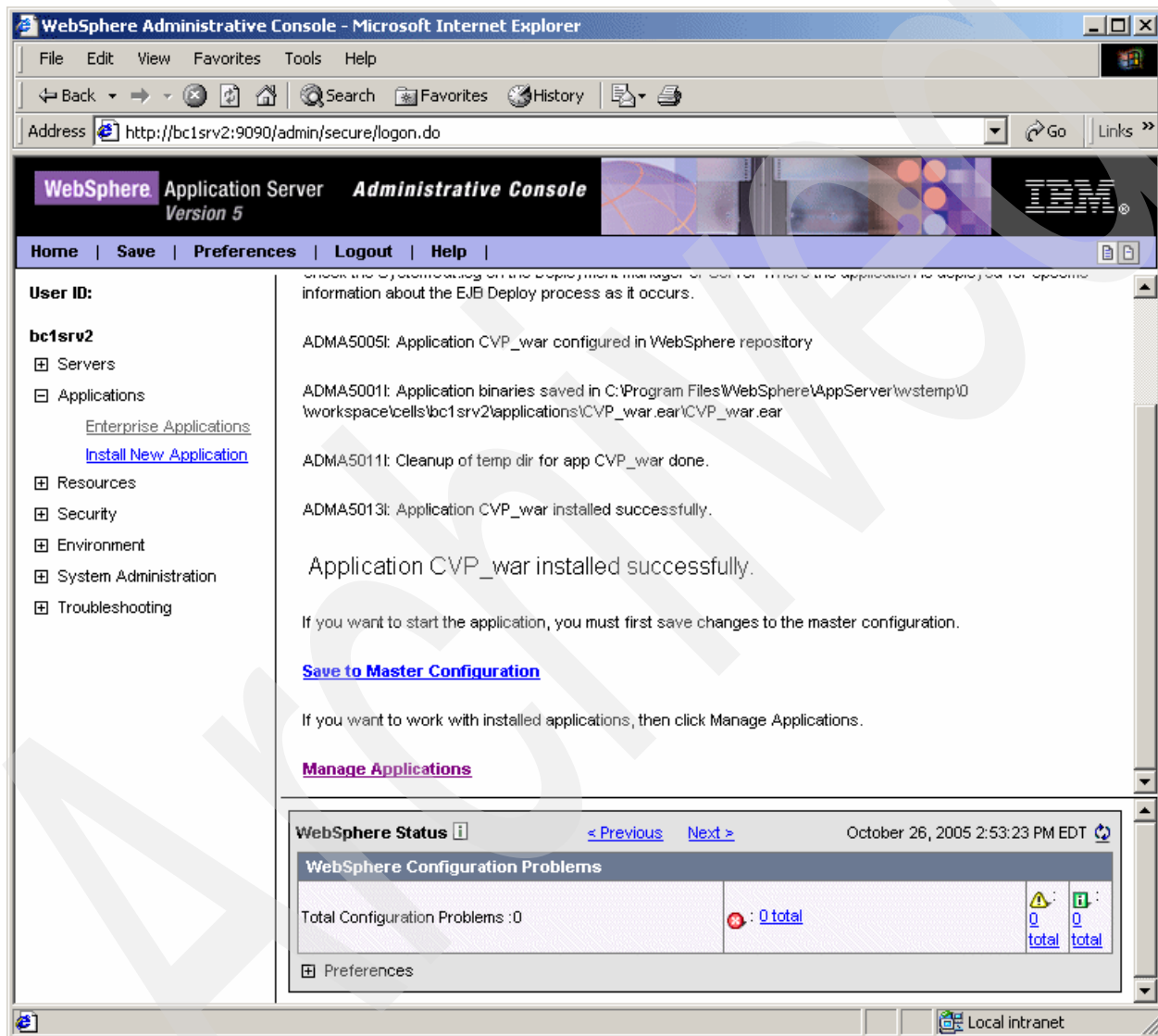


Figure 2-24 Successful installation

Finally, click **Save** so the master repository will be updated with the new application.

### Step 6. WebSphere Variables

The last step is to configure an environment variable. This needs to be set before the Cisco CVP Web application can start. Click **New**.

The screen displays the New page where you must enter the variables. Figure 2-25 on page 33 displays this screen. The two fields that need to be updated are shown in Table 2-1.

Table 2-1 Environment variables to configure

Field Name	Variable
Name	AUDIUM_HOME
Value	INSTALLATION_PATH Default: C:\Cisco\CVP\Server

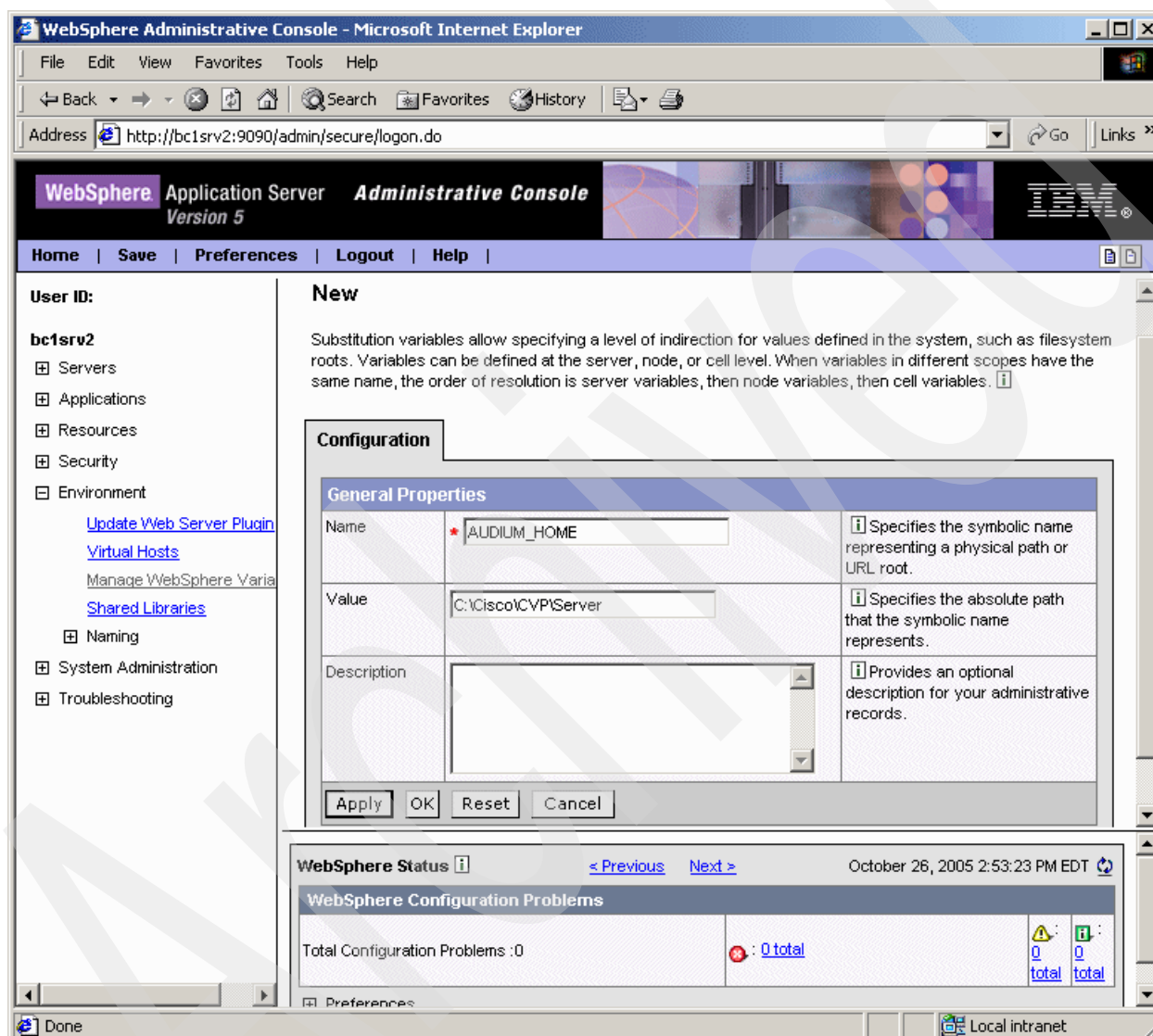


Figure 2-25 Environment variables

1. Click **Apply**.
2. On the next page click **OK**.
3. Finally, click **Save** on the bottom of the page to save the changes to the master configuration. For these changes to take effect, the system needs to be rebooted.

## Configure the Cisco Gateway for Cisco CVP Standalone Mode

In this section, we perform the Gateway configuration steps which are necessary for executing Cisco CVP Standalone applications.

We begin with the files in flash memory. Example 2-1 shows a partial listing of those files, showing only those which are of interest to us here. Appendix A, “Cisco Router Configurations” on page 347 contains the complete list.

*Example 2-1 Partial listing of files in flash memory on the Cisco Gateway*

---

```
1751#dir
Directory of flash:/
. . .
 2 -rw-      17882588 Oct 31 2005 16:07:49 +00:00 c1700-ipvoicek9-mz.124-3a.bin
 3 -rw-         6367 Oct 31 2005 17:04:19 +00:00 CVPSelfService.tcl
 4 -rw-         4830 Oct 31 2005 17:04:46 +00:00 CVPSelfServiceBootstrap.vxml
 5 -rw-         52062 Oct 31 2005 17:05:14 +00:00 error.wav
. . .

33292284 bytes total (15331916 bytes free)
1751#
```

---

The following is a description of each of the files showing in the partial directory listing:

- ▶ **c1700-ipvoicek9-mz.124-3a.bin**  
This file is the IOS image. As you can see by the image name, we are using IOS V12.4(3a), which is the version that is currently recommended by Cisco for use with Cisco CVP V3.1. IOS versions change often. Always check with Cisco for the current recommendation.
- ▶ **CVPSelfService.tcl and CVPSelfServiceBootstrap.vxml**  
These script files must be downloaded from the CVP V3.1 VoiceXML CD. They work together to invoke VoiceXML Server applications. We will see later that each VoiceXML Server application must have a corresponding Application Service configured in the gateway. Each such Application Service configuration consists of a number of application-specific parameters and a pointer to the file, CVPSelfService.tcl. When the correct dialed number arrives, the corresponding Application Service is initiated, which invokes CVPSelfService.tcl based on the parameters provided. CVPSelfService.tcl does little more than collect the gateway's unique call identifier (the H.323 Conference ID), and pass it along to CVPSelfServiceBootstrap.vxml. This action starts a voice browser session. CVPSelfServiceBootstrap.vxml then issues a VoiceXML subdialog tag to invoke the specified application on the VoiceXML Server.
- ▶ **error.wav**  
This file is a recorded error message. It says, "We are experiencing technical difficulties..." and asks the caller to try again later. If you hear this message, it means that the gateway was not able to reach either the primary or the backup VoiceXML Server.

Now we take a look at the gateway's configuration. Appendix A, “Cisco Router Configurations” on page 347 contains a copy of the entire Cisco 1751-V Modular Access Router configuration (obtained by typing **show running-config** at the command line). That configuration contains all items necessary to run both an ICM Integrated application and several CVP Standalone applications. Here, we will present excerpts and discuss the individual items that are relevant to CVP Standalone applications.

**Note:** These excerpts do not necessarily appear here in the same order as they do in the running-config output.

The commands show in Example 2-2 ensure that when debug logs are written, timestamps are written at the millisecond level. Also, only critical error messages are logged to the serial

console port. More verbose logging on that port can seriously degrade gateway performance.

*Example 2-2 Commands that affect debug logs*

---

```
service timestamps debug datetime msec
service timestamps log datetime msec
logging console critical
voice call debug full-guid
```

---

The final line in Example 2-2 on page 35 ensures that all logging that is call-specific also includes the 36-byte global unique identifier (GUID), which the gateway assigns to every call and call leg. This information is vital for debugging call flow failures.

None of these commands actually turn on logging. That is done as needed using the debug command.

You have to tell the gateway that it *is* a gateway, and you also have to tell it that it supports H.323 (see Example 2-3).

*Example 2-3 Command to tell the gateway that it is a Gateway*

---

```
voice service voip
 h323
!
gateway
```

---

The Cisco CVP VoiceXML Server and the Cisco CVP Call Control Server generate some unusual VoiceXML code in order to work around a shortcoming in the VoiceXML V2.0 specification. Suffice it to say, that occasionally they have to attempt to transfer the call to a dummy phone number that is guaranteed to fail. The commands shown in Example 2-4 are necessary so that you cannot accidentally configure a dial-peer on the gateway that matches that dummy phone number. Without these steps, there is a real danger that every call will end by dialing 9 and trying to transfer to an outside phone number! We will later see an additional command that must be added to all VoIP dial-peers.

*Example 2-4 Commands to prevent accidental configuration of a dial-peer that matches a dummy phone number*

---

```
voice translation-rule 1
 rule 1 /987654/ //
```

!

```
voice translation-profile block
 translate called 1
!
```

```
dial-peer voice 987654 voip
 translation-profile incoming block
 incoming called-number 987654
```

---

Example 2-5 shows where we direct the gateway to the MRCP server. If you are using the WebSphere Voice Server Load Balancer, you would include the Load Balancer's cluster address here. Otherwise, you would include the IP address of the Voice Server itself.

**Note:** You can direct ASR and TTS to different servers if desired, but you can only select one of each. If you are supporting any significant call volume at all, it is strongly recommended that you use a Cisco Content Services Switch (CSS), or the WebSphere Voice Server Load Balancer to distribute work to multiple ASR and TTS servers

---

*Example 2-5 Commands to direct the gateway to the MRCP server*

---

```
ivr asr-server rtsp://9.42.171.98/media/recognizer
ivr tts-server rtsp://9.42.171.98/media/synthesizer
```

---

The Application Service (see Example 2-6) is shared by all VoiceXML Server applications. Later, we will create another Application Service that is needed for each individual application.

---

*Example 2-6 Application Service*

---

```
application
service CVPSelfService flash:CVPSelfServiceBootstrap.vxml
  paramspace english language en
  paramspace english index 0
  paramspace english location flash:
  paramspace english prefix en
```

---

**Note:** Whenever you enter or modify entries in the Application Service section of the configuration, you need to instruct the gateway to reload the application. This is done after exiting from configuration mode, by entering the command:

```
call application voice load name
```

The variable, *name*, represents each Application Service you have modified. Applications also must be loaded if you modify the flash memory files to which they point.

The command show in Example 2-7 is important for some ASR servers, and harmless for others. It has to do with establishing the IP address for RTP traffic when load balancers are involved.

---

*Example 2-7 Command dealing with RTP traffic when load balancers are involved*

---

```
mrpc client rtpsetup enable
```

---

The command show in Example 2-8 prevents the gateway from establishing persistent HTTP connections. Testing has shown a certain amount of performance degradation when persistence is enabled.

---

*Example 2-8 Command to prevent the gateway from establishing persistent HTTP connections*

---

```
no http client connection persistent
```

---

First, note the **ip route-cache** command as shown in Example 2-9. This can be present or not, but what you should *never* see is **no ip route-cache**. Route caching provides a very important performance improvement in production deployments.

---

*Example 2-9 ip route-cache command*

---

```
interface FastEthernet0/0
...
ip route-cache same-interface
...
duplex full
speed 100
...
```

---

It is very important that all ethernet interfaces, not only in the gateways and gatekeepers, but also in the Windows machines and any layer two switches in your network, be set to a fixed 100 Mbps data rate, and full-duplex. DO NOT allow autosensing for these parameters. The two commands `duplex full` and `speed 100` accomplish this for IOS devices.

If you do not set these parameters, you might have no trouble at all. Often, however, this can be the source of unexplained intermittent IP transmission delays in your network, which you might not even notice until you are running under load.

The commands in Example 2-10 turn on layer 3 IP routing capabilities in the gateway and identify 9.42.171.3 as the default gateway for IP routing purposes. If you don't include these lines, you may find that media streams to the ASR and TTS servers do not get set up properly.

*Example 2-10 Turn on layer 3 IP routing capabilities*

---

```
ip classless
ip route 0.0.0.0 0.0.0.0 9.42.171.3
```

---

This concludes the basic gateway configuration necessary for Cisco CVP Standalone deployments. There will be further instructions in Chapter 3, “Advanced deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3” on page 47 when we need to configure individual applications. Also, in Section 3.2.3, “Initial Gateway and Gatekeeper Configuration for ICM Integrated Deployments” on page 66, we will discuss the additional IOS configuration necessary in order to run ICM Integrated applications.

**Note:** After you make substantial gateway or gatekeeper configuration changes, you should always enter the command `write memory` in order to save your new running-config to nonvolatile memory.

## 2.4 Installing WebSphere Voice Server V5.1.3

This section describes the process of installing WebSphere Voice Server V5.1.3 on the Microsoft Windows Server 2003 Enterprise Edition operating system. Selection of multiple languages on each server is a new feature of this release, which is covered in this section. At the end of the installation process, the FirstSteps GUI is used to verify that the installation is complete.

The installation process is done using software that is downloaded from the IBM PartnerWorld® Software Access Catalog, which is the normal process used by IBM Business Partners. PartnerWorld is a worldwide program for IBM Business Partners that offers sales and marketing tools, skill-building courses, and technical support to help create opportunities to grow their business and drive increased profit. Partners can self-register with PartnerWorld to obtain access to IBM product information. They can also purchase additional entitlements such as downloading IBM software products through the IBM Software Access Catalog.

The following topics are covered:

- ▶ Obtain the software from the IBM Software Access Catalog
- ▶ Prepare the systems and network operating systems
- ▶ Run the installation program
- ▶ Verify the installation

## 2.4.1 Obtain the software from the IBM Software Access Catalog

WebSphere Voice Server V5.1.3 can be downloaded as multiple CD images packaged as eAssemblies. Each eAssembly contains product images for one or more operating system platforms. Images are packaged as TAR, ZIP, or self-extracting executable EXE files. WebSphere Voice Server V5.1.3 is divided into three eAssemblies. The base product for multiplatforms, language support for Automatic Speech Recognition (ASR) and Concatenative Text to Speech (CTTS). Languages for WebSphere Voice Server V5.1.3 are available for both the AIX, Linux, and Windows Server 2003 operating systems. For this Redpaper we will use Microsoft Windows Server 2003.

### Available versions of the WebSphere Voice Server:

The following steps show you how to list available versions of WebSphere Voice Server.

1. Begin the process of downloading WebSphere Voice Server by visiting the following URL:  
<http://www.developer.ibm.com/welcome/softmall.html>
2. Click **Log in to Software Access Catalog**.
3. Enter your PartnerWorld user ID and password, if requested.
4. Click **Yes** if you receive any security warnings.
5. Read the IBM PartnerWorld Agreement and select **I Agree** at the bottom of the screen.
6. Click **Software Access Catalog - Electronic Software Download**.
7. Use option 3 **Search Product Descriptions** and enter WebSphere Voice Server V5.1.3 leaving the default **Exact search** option.
8. Click **Search**.
9. You are presented with a list of all WebSphere Voice Server V5.1.3 products. Notice the entries under the heading Multi-File Assemblies for V5.1.3

### Download WebSphere Voice Server base code

The WebSphere Voice Server V5.1.3 base code eAssembly contains the following products and images.

- ▶ WebSphere Voice Server V4.2 (AIX V5.1)
- ▶ WebSphere Voice Server V5.1.3
- ▶ TTS Connector for Genesys GVP for WebSphere Voice Server V5.1.3
- ▶ Voice Toolkit V6.0.1 (requires Rational® Development Platform downloaded separately)
- ▶ WebSphere Application Server V5.1 and Fixes
- ▶ WebSphere Application Server Network Deployment V5.1 Deploy Manager and Fixes
- ▶ WebSphere Application Server Network Deployment V5.1 Edge Components and Fixes

**Attention:** WebSphere Voice Server V5.1.3, WebSphere Application Server V5.1, and WebSphere Application Server V5.1 Fixes are the minimum required downloads. WebSphere Application Server Network Deployment V5.1 (Edge Components and Deploy Manager) are used for advanced, multi-server installations. The Voice Toolkit V6.0.1 is used separately for voice application development and debugging.

### Display a list of the product images in the base code eAssembly

To display of list of images, perform the following steps.

1. Click **WebSphere Voice Server V5.1.3 Multiplatforms**.
2. View and read the Overall Program License Agreement and click **Accept Terms**.



- Each product image listed gives you the option to Initiate Download with Restartable Transfer or HTTP transfer at a variety of download sites. Restartable Transfer is quicker because it uses an applet to open multiple connections. Using this option, you can queue multiple file downloads by clicking **Previous Page** (bottom of page) and selecting another product image while the previous file is downloading. At a minimum, download WebSphere Voice Server V5.1.3, WebSphere Application Server V5.1, and WebSphere Application Server V5.1 Fixes for the operating system you are using by clicking **Accept Terms and Download** under the desired product image. Place each downloaded ZIP or EXE file in a separate directory on your local machine.

Windows system images include ZIP files, which can be expanded using your favorite unzip utility and self-extracting executables.

- You can remove the download file after expansion. If you elect to first burn a CD from the downloaded images, be sure to use the root of the ZIP image as the root of your CD. In other words, do not include any directory names that you created to download the image on the CD. The WebSphere Voice Server Installation Wizard prompts you for each image location so you can either specify the download directory or your CD drive location.

### Download WebSphere Voice Server language images

To display a list of the product images in the Language Support (ASR) and Concatenative Text to Speech (CTTS) eAssemblies, perform the following steps:

- Click the link for the desired ASR or CTTS. For example, **WebSphere Voice Server Multiplatforms V5.1.3 US English Language Support**.
- View and read the Overall Program License Agreement and click **Accept Terms**.
- Download and expand the file for desired operating system.

## 2.4.2 Prepare the systems and network operating systems

WebSphere Voice Server requires a static IP address. Configure your operating system network settings to use a static IP address and confirm that the hosts file is set up correctly. The hosts file is in the C:\Windows\system32\drivers\etc\hosts directory on Windows. Example 2-11 shows the HOSTS file of the Voice Server used in the basic deployment.

#### *Example 2-11 HOSTS file example*

---

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
9.42.171.98 bc1srv8.itso.ral.ibm.com bc1srv8
127.0.0.1 localhost.localdomain localhost
```

---

Supported hardware platforms for WebSphere Voice Server V5.1.3 on Windows systems that meet the minimum hardware criteria defined here:

- ▶ Processor: x86 1 GHz (minimum)
- ▶ Memory (RAM): 2 GB (minimum)
- ▶ Available Disk Space: 2 GB
- ▶ Network: TCP/IP
- ▶ Other: CD-ROM

See the WebSphere Voice Server Information Center for the current information.

<http://publib.boulder.ibm.com/infocenter/pvcvoice/51x/index.jsp>

**Restriction:** No Windows Service Packs are supported at this time.

### 2.4.3 Run the installation using LaunchPad

The installation steps included here are unique to the software download approach used by IBM Business Partners.

Installation instructions that are CD-based are available in the WebSphere Voice Server database:

<http://publib.boulder.ibm.com/infocenter/pvcvoice/51x/index.jsp>

**Important:** WebSphere Application Server and all required fix packs are installed silently during the installation process. The WebSphere Application Server fix pack provided with WebSphere Voice Server V5.1.3 must be used as it has been tailored specifically for this installation. It is recommended that you allow this default method to be used rather than run a separate installation the first time.

1. Use Information Explorer to locate the folder that contains the installation software. In this case, locate the directory for the **LaunchPad.exe** file and double-click it. This action launches the installation wizard shown in Figure 2-26.

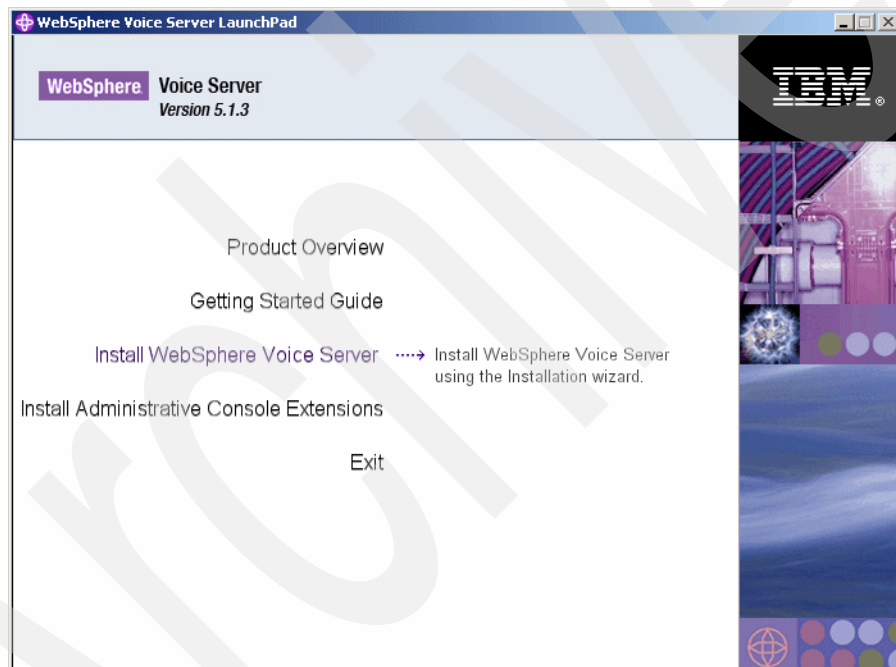


Figure 2-26 LaunchPad Welcome panel

2. Click **Install WebSphere Voice Server** to continue.
3. The system displays an Install program has been launched message. Click **OK**.
4. In the InstallShield Welcome panel, click **Next** to continue.
5. Click **I accept the terms in the license agreement** message if you agree to the license agreement and click **Next** to continue. After you accept the licensing terms, the installation wizard checks for prerequisites and for previous versions.
6. The Setup screen prompts you to select either a Full or Custom install as shown in Figure 2-27 on page 41.

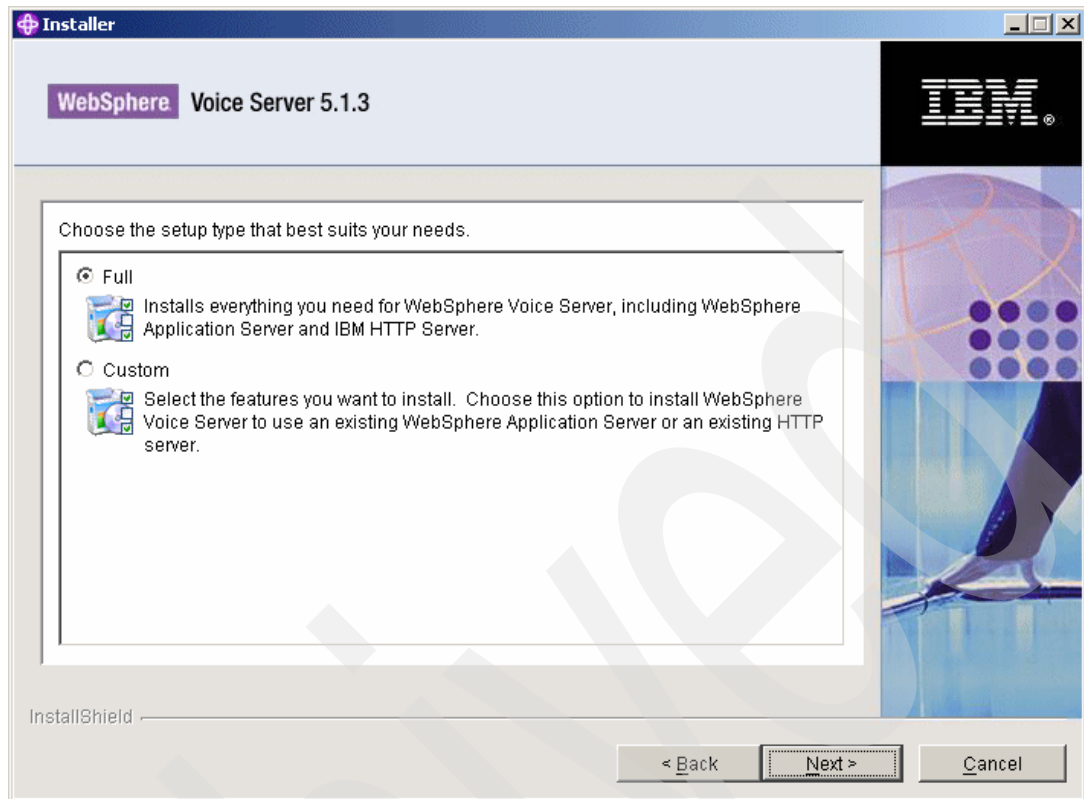


Figure 2-27 Choose the Setup Type

7. Click **Full** to install WebSphere Application Server and WebSphere Voice Server. Click **Next** to continue.
8. Because you are installing WebSphere Application Server for the first time, you are prompted to choose installation directories for WebSphere Application Server and for the IBM HTTP Server. Accept the default directories for each.
9. Configure both WebSphere Application Server and IBM HTTP Server as a service as shown in Figure 2-28 on page 42.

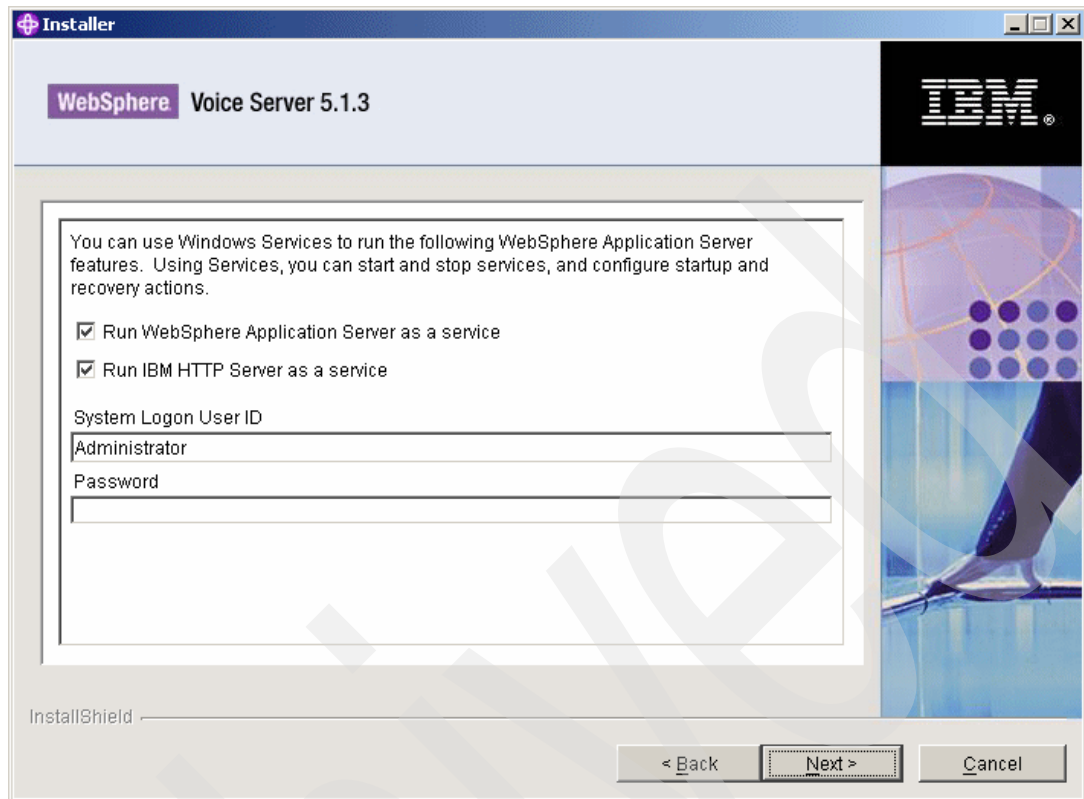


Figure 2-28 Use Windows Services

10. Enter a logon user ID and a password. Make sure that the logon user ID belongs to the Administrator group and has the following advanced user rights:

- Act as part of the operating system.
- Log on as a service.

**Important:** When installing WebSphere Application Server as a Windows service, do not use a user ID that contains spaces. A user ID with spaces cannot be validated and the user is not allowed to proceed with the installation.

11. Check and accept the default node name and a host name for WebSphere Application Server. If an incorrect location for WebSphere Application Server is displayed, enter the correct location in the field at the bottom of the panel and click **Next** to continue.

12. The installation wizard prompts you for the WebSphere Application Server fix pack CD to install the required WebSphere Application Server fix packs. Click **Browse** and locate the correct directory and then click **Next** to continue.

**Tip:** When browsing to locate the correct directory, if the installation program does not accept the directory you choose, try going up one directory level. When selecting TTS folders we found this tip to be useful.

13. In the same way, choose the destination directory for WebSphere Voice Server. Click **Next** to continue.

14. Next, the language panel for WebSphere Voice Server displays. Select the WebSphere Voice Server TTS and ASR capabilities that you require as shown in Figure 2-29.

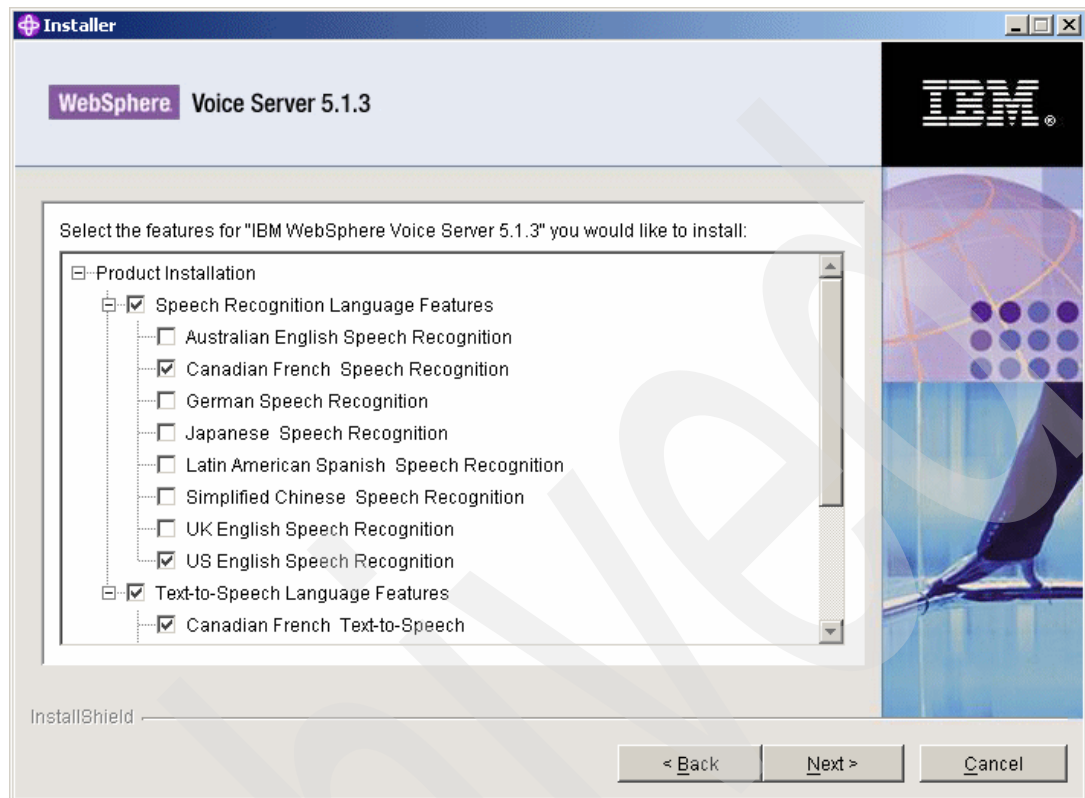


Figure 2-29 Select features

15. Click **Next** to continue. A summary panel displays your install selections.
16. Click **Next** to start the installation process.
17. You are prompted to provide various image locations for each component.
18. A summary panel of the installation with instructions to reboot opens, as shown in Figure 2-30 on page 44.

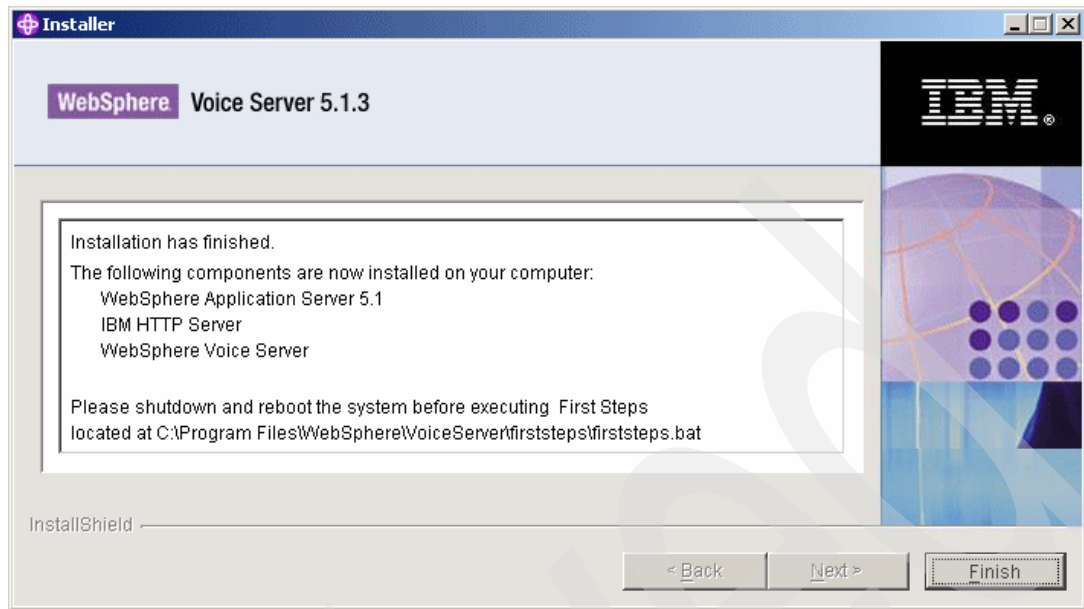


Figure 2-30 Installation has finished

19. Click **Finish** to close the installation wizard. The Welcome panel opens. Click **Exit**.

**Tip:** If you receive any warnings during installation, examine the %TEMP%\CWVOptInstall.log file to verify that there were no file system problems, or other unusual errors. If there are problems, correct them, uninstall the product, and then reinstall.

## 2.4.4 Verify the installation

To verify the installation, follow these steps:

1. If you have not already done so, Reboot the system.
2. Run the **FirstSteps** command from the **Windows Start** → **Programs** → **IBM WebSphere** → **Voice Server V5.1.3** → **First Steps** menu. See Figure 2-31.

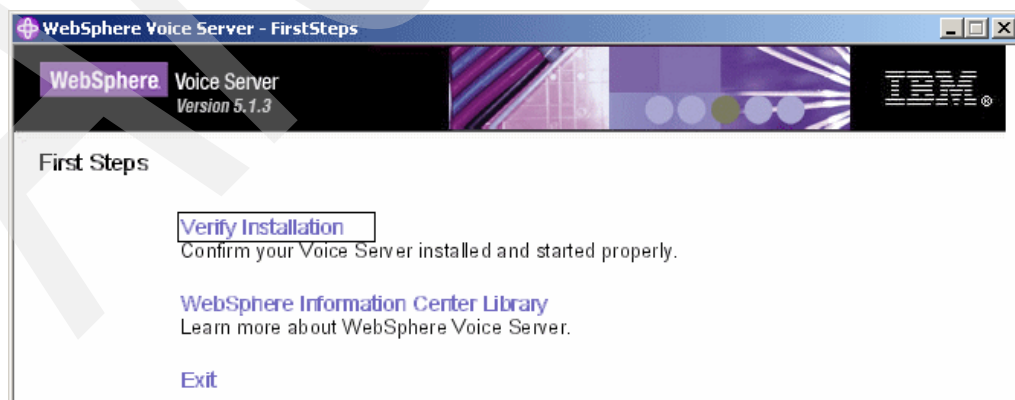
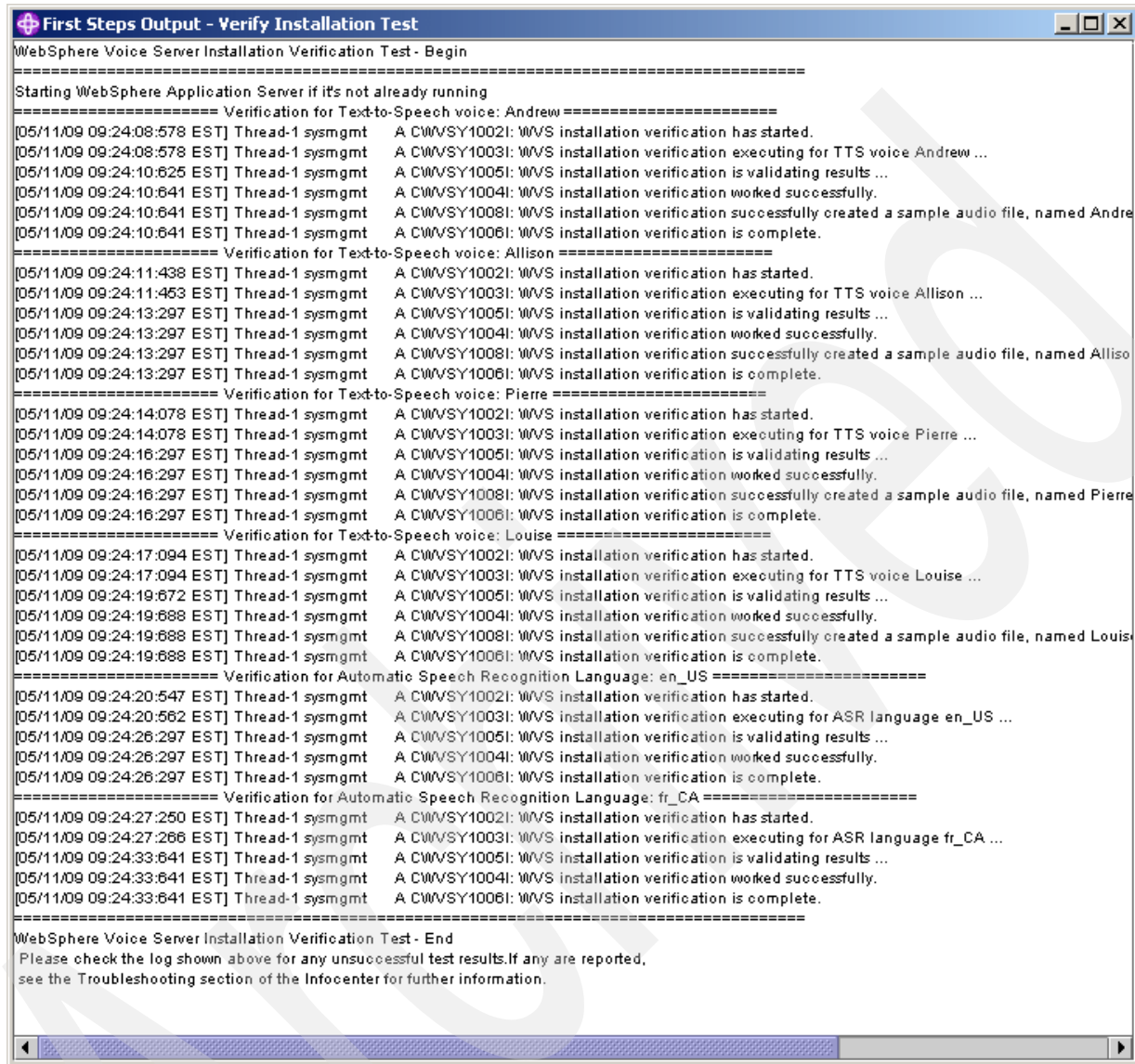


Figure 2-31 WebSphere Voice Server FirstSteps

3. Select **Verify Installation**.



4. This will take several minutes and should result in verification of the recognition and text to speech capabilities as shown in Figure 2-32.



```
First Steps Output - Verify Installation Test
WebSphere Voice Server Installation Verification Test - Begin

Starting WebSphere Application Server if it's not already running

===== Verification for Text-to-Speech voice: Andrew =====
[05/11/09 09:24:08:578 EST] Thread-1 sysmgmt A CWVSY1002I: WVS installation verification has started.
[05/11/09 09:24:08:578 EST] Thread-1 sysmgmt A CWVSY1003I: WVS installation verification executing for TTS voice Andrew ...
[05/11/09 09:24:10:625 EST] Thread-1 sysmgmt A CWVSY1005I: WVS installation verification is validating results ...
[05/11/09 09:24:10:641 EST] Thread-1 sysmgmt A CWVSY1004I: WVS installation verification worked successfully.
[05/11/09 09:24:10:641 EST] Thread-1 sysmgmt A CWVSY1008I: WVS installation verification successfully created a sample audio file, named Andre
[05/11/09 09:24:10:641 EST] Thread-1 sysmgmt A CWVSY1006I: WVS installation verification is complete.

===== Verification for Text-to-Speech voice: Allison =====
[05/11/09 09:24:11:438 EST] Thread-1 sysmgmt A CWVSY1002I: WVS installation verification has started.
[05/11/09 09:24:11:463 EST] Thread-1 sysmgmt A CWVSY1003I: WVS installation verification executing for TTS voice Allison ...
[05/11/09 09:24:13:297 EST] Thread-1 sysmgmt A CWVSY1005I: WVS installation verification is validating results ...
[05/11/09 09:24:13:297 EST] Thread-1 sysmgmt A CWVSY1004I: WVS installation verification worked successfully.
[05/11/09 09:24:13:297 EST] Thread-1 sysmgmt A CWVSY1008I: WVS installation verification successfully created a sample audio file, named Alliso
[05/11/09 09:24:13:297 EST] Thread-1 sysmgmt A CWVSY1006I: WVS installation verification is complete.

===== Verification for Text-to-Speech voice: Pierre =====
[05/11/09 09:24:14:078 EST] Thread-1 sysmgmt A CWVSY1002I: WVS installation verification has started.
[05/11/09 09:24:14:078 EST] Thread-1 sysmgmt A CWVSY1003I: WVS installation verification executing for TTS voice Pierre ...
[05/11/09 09:24:16:297 EST] Thread-1 sysmgmt A CWVSY1005I: WVS installation verification is validating results ...
[05/11/09 09:24:16:297 EST] Thread-1 sysmgmt A CWVSY1004I: WVS installation verification worked successfully.
[05/11/09 09:24:16:297 EST] Thread-1 sysmgmt A CWVSY1008I: WVS installation verification successfully created a sample audio file, named Pierre
[05/11/09 09:24:16:297 EST] Thread-1 sysmgmt A CWVSY1006I: WVS installation verification is complete.

===== Verification for Text-to-Speech voice: Louise =====
[05/11/09 09:24:17:094 EST] Thread-1 sysmgmt A CWVSY1002I: WVS installation verification has started.
[05/11/09 09:24:17:094 EST] Thread-1 sysmgmt A CWVSY1003I: WVS installation verification executing for TTS voice Louise ...
[05/11/09 09:24:19:672 EST] Thread-1 sysmgmt A CWVSY1005I: WVS installation verification is validating results ...
[05/11/09 09:24:19:688 EST] Thread-1 sysmgmt A CWVSY1004I: WVS installation verification worked successfully.
[05/11/09 09:24:19:688 EST] Thread-1 sysmgmt A CWVSY1008I: WVS installation verification successfully created a sample audio file, named Louis
[05/11/09 09:24:19:688 EST] Thread-1 sysmgmt A CWVSY1006I: WVS installation verification is complete.

===== Verification for Automatic Speech Recognition Language: en_US =====
[05/11/09 09:24:20:547 EST] Thread-1 sysmgmt A CWVSY1002I: WVS installation verification has started.
[05/11/09 09:24:20:562 EST] Thread-1 sysmgmt A CWVSY1003I: WVS installation verification executing for ASR language en_US ...
[05/11/09 09:24:26:297 EST] Thread-1 sysmgmt A CWVSY1005I: WVS installation verification is validating results ...
[05/11/09 09:24:26:297 EST] Thread-1 sysmgmt A CWVSY1004I: WVS installation verification worked successfully.
[05/11/09 09:24:26:297 EST] Thread-1 sysmgmt A CWVSY1006I: WVS installation verification is complete.

===== Verification for Automatic Speech Recognition Language: fr_CA =====
[05/11/09 09:24:27:250 EST] Thread-1 sysmgmt A CWVSY1002I: WVS installation verification has started.
[05/11/09 09:24:27:266 EST] Thread-1 sysmgmt A CWVSY1003I: WVS installation verification executing for ASR language fr_CA ...
[05/11/09 09:24:33:641 EST] Thread-1 sysmgmt A CWVSY1005I: WVS installation verification is validating results ...
[05/11/09 09:24:33:641 EST] Thread-1 sysmgmt A CWVSY1004I: WVS installation verification worked successfully.
[05/11/09 09:24:33:641 EST] Thread-1 sysmgmt A CWVSY1006I: WVS installation verification is complete.

WebSphere Voice Server Installation Verification Test - End
Please check the log shown above for any unsuccessful test results.If any are reported,
see the Troubleshooting section of the Infocenter for further information.
```

Figure 2-32 Installation Verification Test Result

The installation of WebSphere Voice Server includes configuration tuning for WebSphere Application Server and your operating system. To see tips on additional ways to improve the performance of your WebSphere Voice Server, refer to the topic on *Tuning* in the WebSphere Voice Server Information Center:

<http://publib.boulder.ibm.com/infocenter/pvcvoice/51x/index.jsp>

**Tip:** For autostart of WebSphere Voice Server V5.1.x please refer to *IBM WebSphere Voice Server for Multiplatforms V5.1.1/V5.1.2 Handbook, SG24-6447*.

## 2.4.5 Network infrastructure for basic deployment

After the Cisco router, Cisco CVP V3.1, and WebSphere Voice Server for Multiplatforms V5.1.3 have been installed and configured, the basic environment should now be ready for testing applications. The Redpaper physical topology can be seen in Figure 2-33.

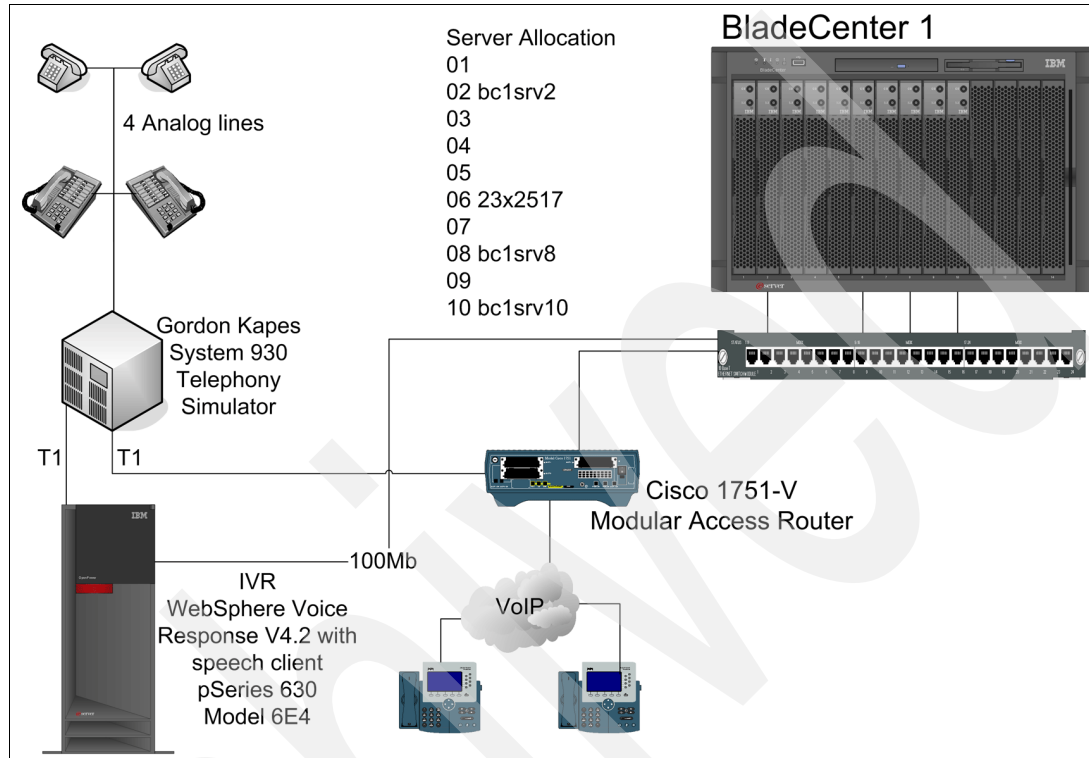


Figure 2-33 Network infrastructure for basic deployment

The IP address settings of all the components are displayed in Table 2-2.

Table 2-2 IP addresses in basic deployment

Function	Host name	IP address
Cisco CVP V3.1 VoiceXML Server with Tomcat	bc1srv10	9.42.171.24
Cisco CVP V3.1 VoiceXML Server with WebSphere Application Server V5.1.1	bc1srv2	9.42.171.94
WebSphere Voice Server for multiplatforms V5.1.3	bc1srv8	9.42.171.98
Cisco 1751-V Modular Access Router		9.42.171.131
My SQL server V5.0.1.5	23x2517	9.42.171.150
WebSphere Voice Response with speech client V4.2		9.42.171.16



## **Advanced deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3**

This chapter describes how to deploy a complex IVR solution using Cisco CVP V3.1 and WebSphere Voice Server for multiplatforms V5.1.3. The following topics are discussed:

- ▶ Introduction to solution
- ▶ Overall environment
- ▶ Installation and configuration of additional Cisco CVP V3.1 components
- ▶ Installation and configuration steps for WebSphere Edge Server Load Balancer V5.1.1.41 with multiple voice servers
- ▶ Cisco AS5400HPX Universal Gateway and a Cisco 3660 router setup and configuration.

## 3.1 Complex deployment overview

The basic deployment addressed the need for a development environment. This allowed for application development and testing. However, it could not handle a large amount of concurrent calls at one time. This chapter builds on the basic deployment to create a more robust, resilient solution. The increased workload due to many concurrent call is balanced. The call flow is also tracked and managed with additional functionality enhancements. By no means is it a complete solution. Rather its an example to illustrate how a complex deployment is planned, installed and configured.

The following components were installed on top of the basic deployment:

- ▶ Multiple WebSphere Voice Server V5.1.3 servers
- ▶ WebSphere Edge Server Load Balancer V5.1.1.41
- ▶ Cisco CVP V3.1 Call Control Server
- ▶ Cisco ICM V6.0 SR4
- ▶ Cisco AS5400HPX Universal Gateway
- ▶ Cisco 3660 Gatekeeper

### 3.1.1 Complex deployment solution topology

Figure 3-1 depicts the complex deployment infrastructure used by the Redpaper team. With the addition of Load Balancer, the WebSphere Voice Server V5.1.3 failover, recoverability, maintainability, or serviceability of the Voice Server can be achieved. If one of the machines must be serviced or fails, then your environment will still be available to handle calls.

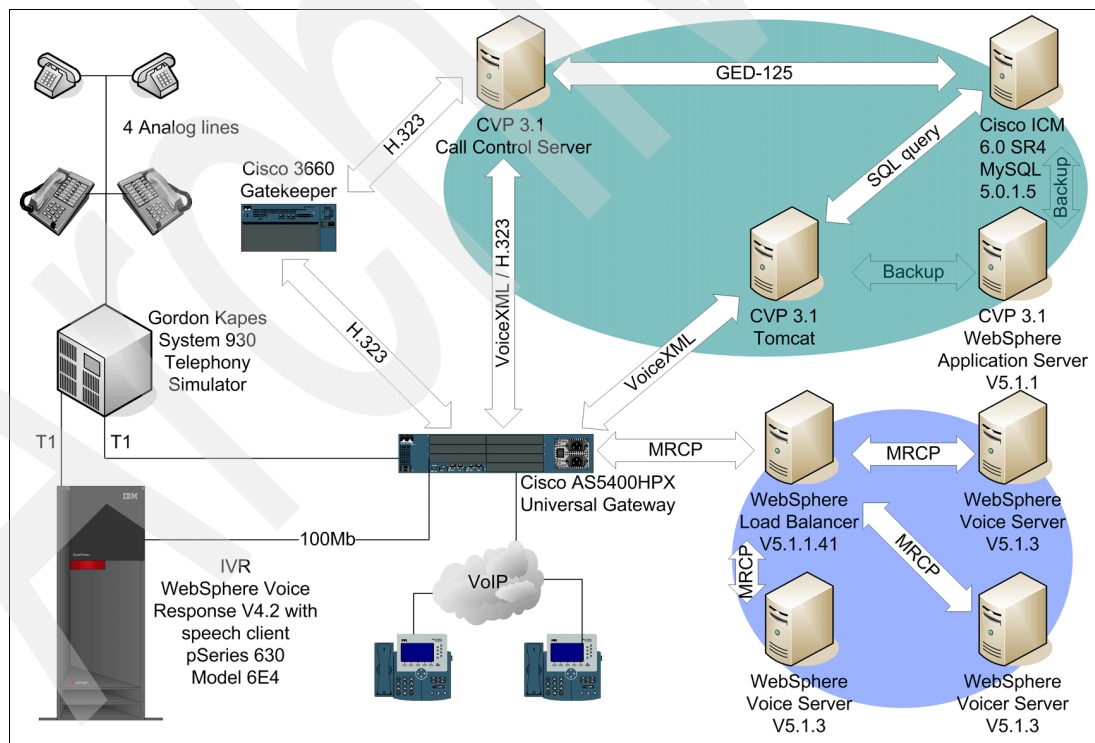


Figure 3-1 Complex deployment of Cisco CVP V3.1 and a Cisco AS5400HPX Universal Gateway

## 3.2 ICM Integrated Deployment

This section describes the process of installing the additional components that are necessary to integrate Cisco CVP V3.1 into an ICM contact center environment. Because this Redpaper focuses on Cisco CVP and its interoperability with IBM WebSphere products, we assume that ICM is already installed and functioning. We do, however, describe ICM configuration to the extent necessary to *add* Cisco CVP integration.

The following topics are covered:

- ▶ Installation of the Cisco CVP V3.1 Call Control Server
- ▶ Initial Configuration of the Cisco CVP V3.1 Call Control Server
- ▶ Initial Gateway and Gatekeeper Configuration
- ▶ ICM Configuration

### 3.2.1 Installation of the Cisco CVP V3.1 Call Control Server

The Cisco CVP Call Control Server can be viewed as the component which connects the Cisco CVP VoiceXML Server to ICM. Insert the Cisco CVP V3.1 Installer CD, and navigate to the *Install* directory.

1. Double-click **Setup.exe**, as shown in Figure 3-2. If you are upgrading from a previous version, select **Setup.exe** from the *Upgrade* directory.

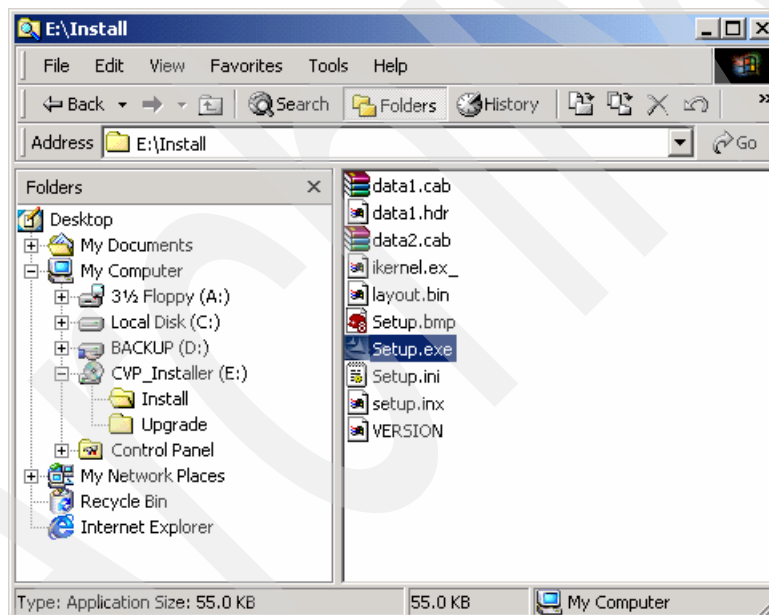


Figure 3-2 Start the Installer

Screen shot reprinted by permission from Microsoft Corporation

2. The installer presents its welcome message, shown in Figure 3-3 on page 50. Ensure that no other programs are running, and click **Next**.

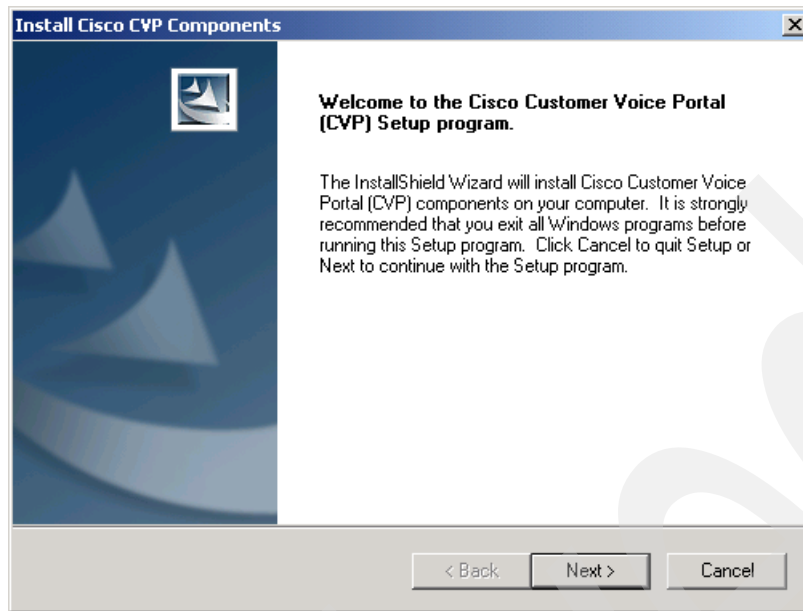


Figure 3-3 Cisco CVP V3.1 Welcome Screen

3. A copyright screen (Figure 3-4 on page 50) is shown. Click **Next**.

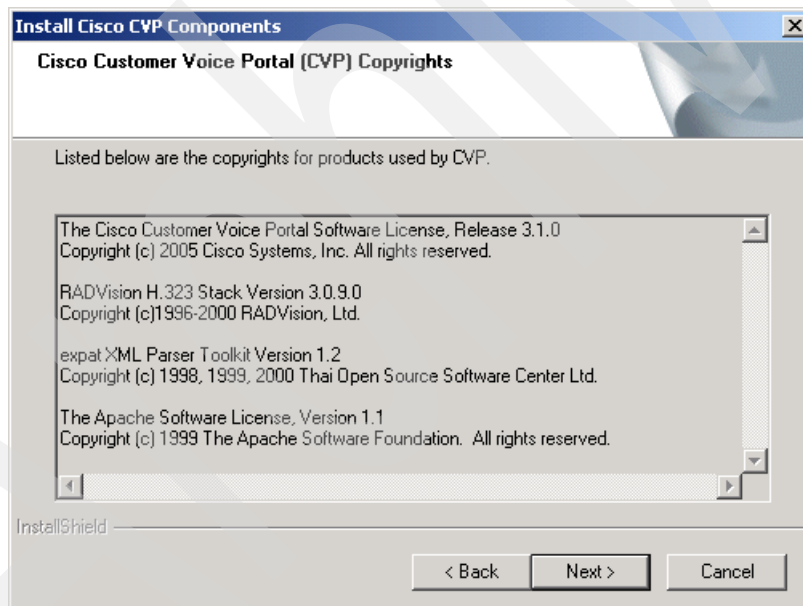


Figure 3-4 Cisco CVP V3.1 Copyright Screen

4. Read and accept the license agreement, shown in Figure 3-5 on page 51, by clicking **Yes**.

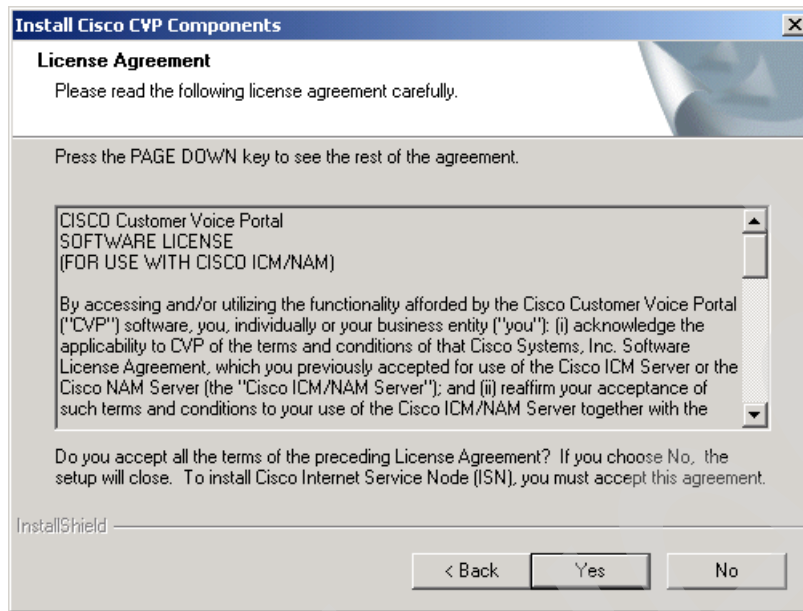


Figure 3-5 Cisco CVP V3.1 License Agreement

5. The Installer then shows the location on disk where Cisco CVP V3.1 will be installed. You can change it to a different directory or a different disk, if appropriate.

The term *Internet Service Node* (ISN) is the name that was used for earlier versions (prior to V3.0) of the Cisco Customer Voice Portal product. In this installer as well as in this document, *ISN* and *CVP* should be considered synonymous.

In this Redpaper we accepted the default location as shown in Figure 3-6 on page 51. Click **Yes**.

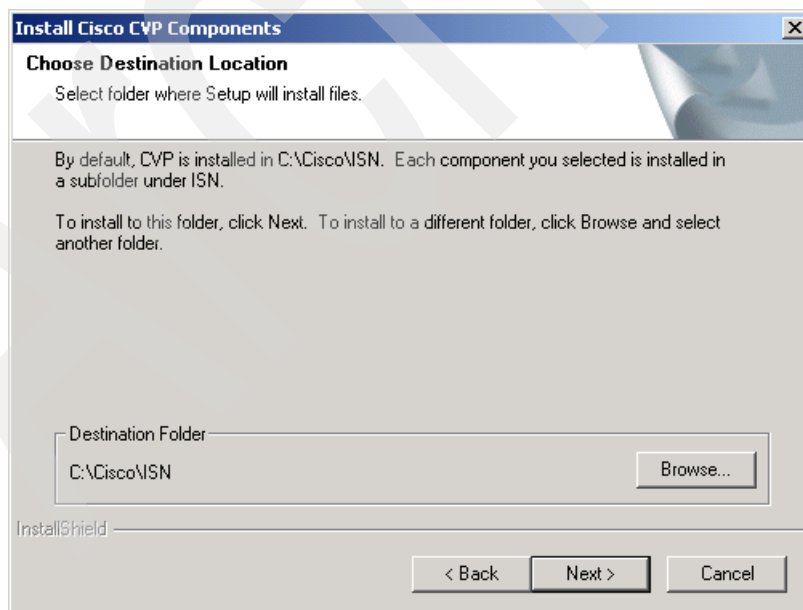


Figure 3-6 Cisco CVP V3.1 Install Destination

6. The Installer then presents a dialog box containing the list of components which can be installed. The default selections, shown in Figure 3-7 on page 52, are satisfactory for our

needs. For this lab setup we do not require the SDDSN components, which are used for sending SNMP traps to a management console, nor do we need the Spanish system media files.

The English system media files would normally be placed on the Media Server. That can be accomplished by allowing the Installer to lay them down here, and then copying the files manually to the Media Server, or by running the Installer directly on the Media Server and deselecting all but the System Media Files check box. Spanish system media files may be installed similarly.

Select the appropriate checkboxes and click **Next**.

**Note:** You will notice that an ICM Integrated Cisco CVP solution contains two kinds of application servers and two kinds of voice browsers. The names are historical and unfortunate. At one time, the components described here were the only application servers and voice browsers in the network, but as of Cisco CVP V3.0 most of those functions have moved to the Cisco CVP VoiceXML Server and the VoiceXML Gateway's Voice Browser, respectively. The older components still perform important activities, but their primary functions are no longer those which their names would imply.

This section refers only to the older components that are installed on the Cisco CVP Call Control Server. We use the names *CVP Application Server* and *CVP Voice Browser* here to distinguish them from their newer counterparts.

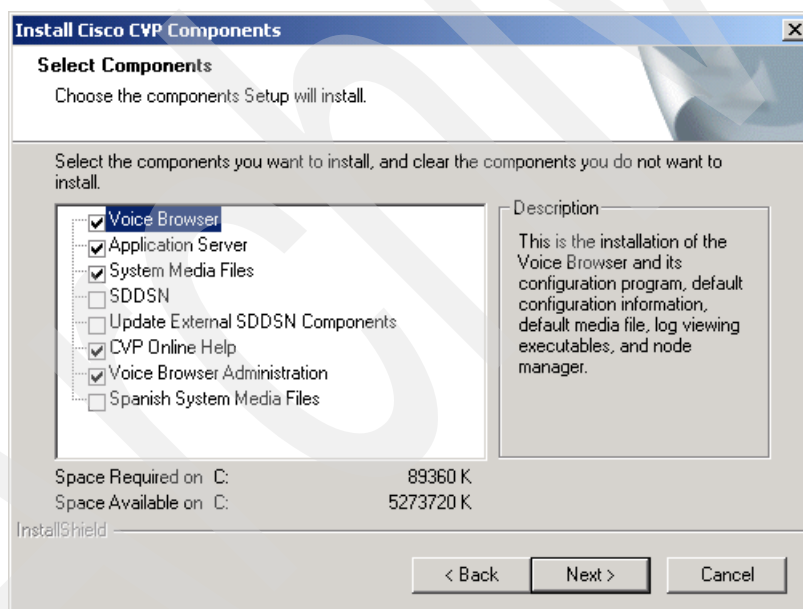


Figure 3-7 Select Cisco CVP V3.1 Components to Install

7. The Installer then provides the opportunity to modify the program folder where Cisco CVP V3.1 will be installed. We accepted the default location, as shown in Figure 3-8 on page 53. Click **Next**.

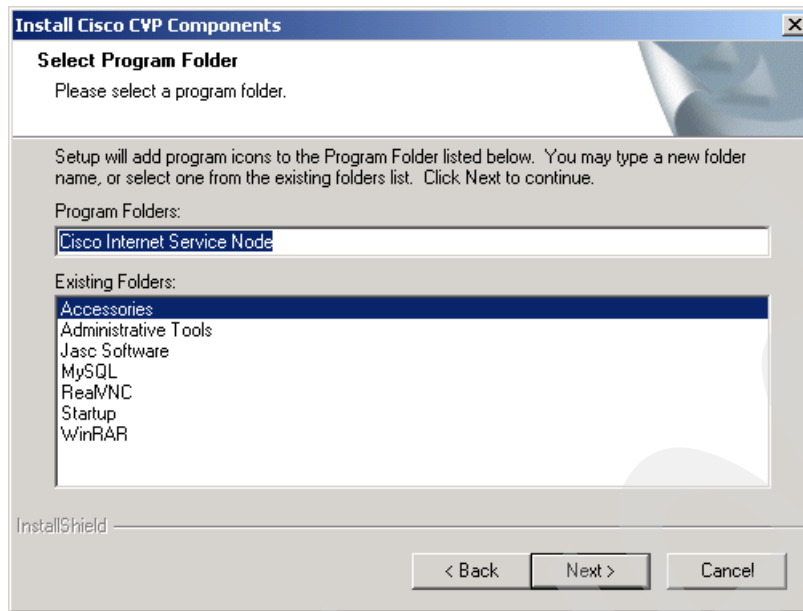


Figure 3-8 Start Menu Program Folder

8. Next, the Installer allows you to select Node Manager properties for Cisco CVP V3.1. The Node Manager is a background program that watches the Cisco CVP Voice Browser process and the Cisco CVP Application Server process. If either fails, the Node Manager automatically restarts it. Node Manager is also responsible for starting these processes when the system boots, and for rebooting the system if a fatal error is detected.

In a production environment you would want the system to automatically start these processes at boot time, and you would want fatal errors to automatically reboot the system. For our lab environment, we accepted the auto-restart, but disabled the reboot on error, as shown in Figure 3-9 on page 53.

Click **Next**.

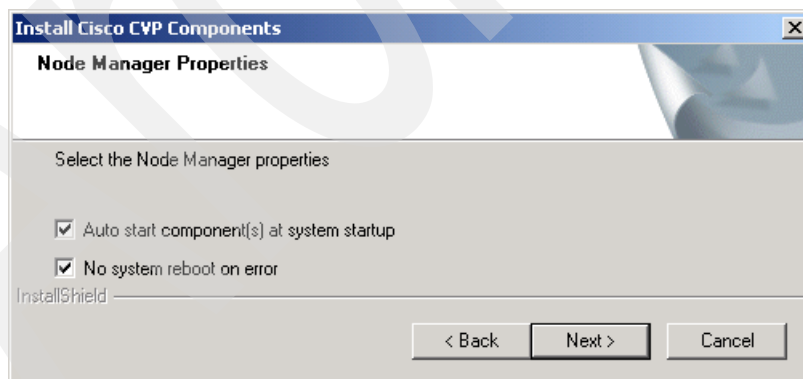


Figure 3-9 Cisco CVP V3.1 Node Manager Properties

9. When Cisco CVP V3.1 runs, several process windows show up minimized in the Windows task bar. You can open these windows and watch various informational and error messages scroll by. These messages are also stored in log files, but sometimes during initial installation and setup it is useful to see them in real time. However, scrolling process windows impose substantial delays on running processes, so in production environments it is absolutely imperative that these windows be kept minimized.

In order to enforce this, the Installer here provides the opportunity to hide the Cisco CVP V3.1 process windows completely, so they do not even appear on the task bar. We are still in the setup phase and not going into production any time soon, so we chose to have the process windows displayed. This is shown in Figure 3-10.

Click **Next**.

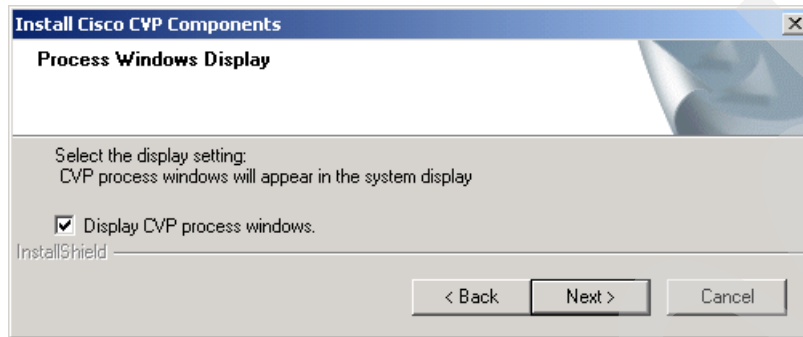


Figure 3-10 Cisco CVP V3.1 Process Windows Setting

10. One of the Cisco CVP V3.1 components, the CVP Application Server, is administered using web pages, and served by Microsoft Internet Information Service (IIS). This administration facility, known as *CVP V3.1 Application Administration*, or *App Admin*, is made up of a set of web pages which must be accessible to IIS. By default, they are stored in IIS's standard web pages directory. We accepted the default location, as shown in Figure 3-11 on page 54.

Click **Next**.

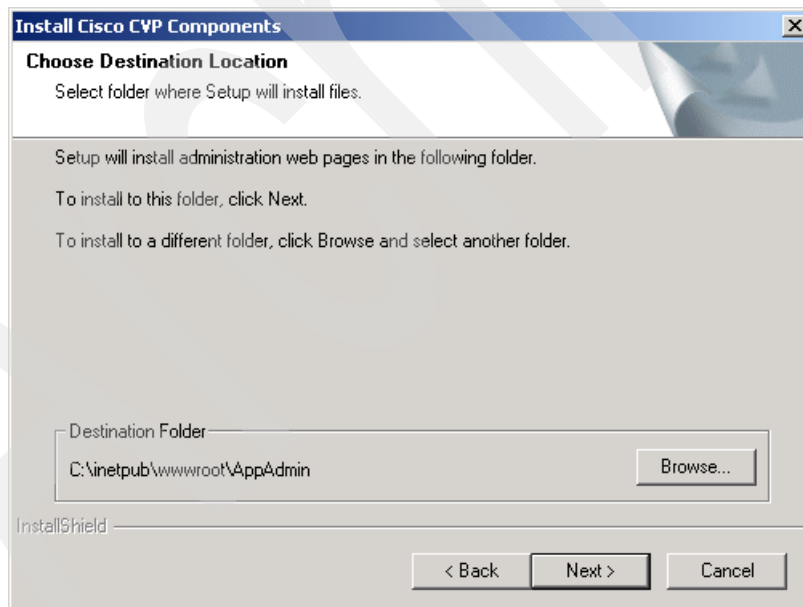


Figure 3-11 Cisco CVP V3.1 Application Server Administration Destination

11. During the installation process, an embedded Lightweight Directory Access Protocol (LDAP) directory server named *DC Directory* is installed. This facility requires local administrator privileges. The following Installer screen therefore requests an appropriate administrator username and password.



It is very important to enter a valid name and password here and, as the dialog box states, when you login again after reboot. The Installer *does not verify its accuracy*. If you make a mistake here, DC Directory will fail to start once you reboot, and the only remedy is to uninstall Cisco CVP V3.1 and begin the process again.

Enter the appropriate username and password as shown in Figure 3-12, and click **Next**.

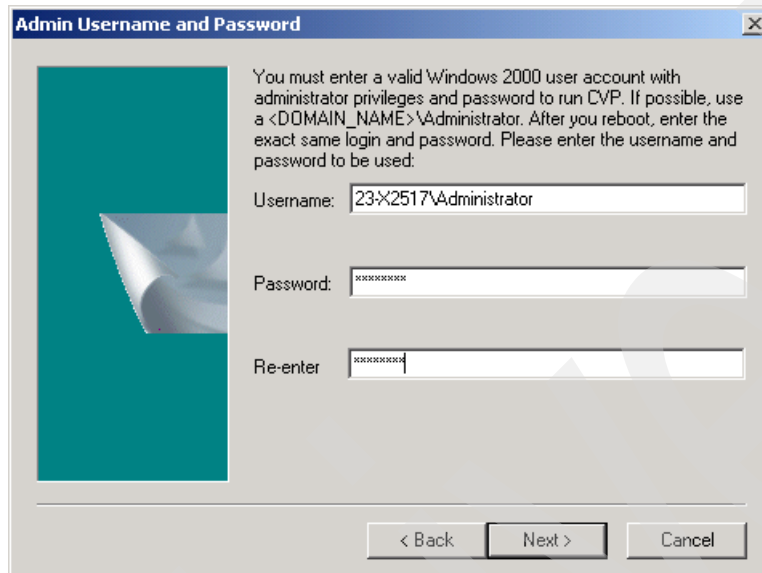


Figure 3-12 Administrator Username and Password

12. At this point the Installer begins its work. Several progress indications appear, and eventually the query shown in Figure 3-13 waits for you to allow it to stop one or more system services.

Press y followed by Enter.

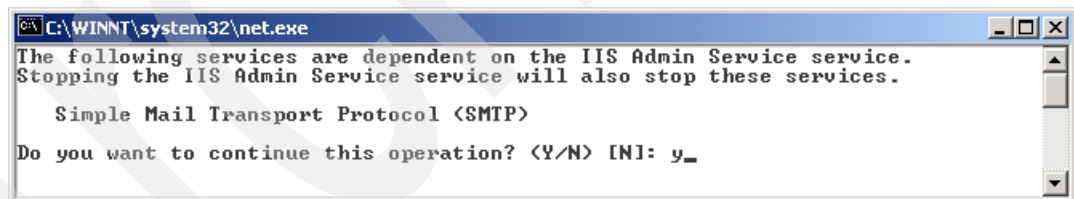


Figure 3-13 Confirmation to Stop SMTP

13. Finally, the Installer is ready to reboot your system. Though the dialog box shown in Figure 3-14 on page 56 seems to be giving you a choice, you *must* reboot at this point. Remove the CD from the drive, click **Yes**, and then click **Finish**.

**Note:** There is a reference to Cisco Security Agent (CSA) on this screen. CSA is a Cisco software product for intrusion detection which has been customized for systems running Cisco CVP V3.1. The CVP V3.1 documentation describes how to acquire it (for free) and install it. For this Redpaper we chose not to use CSA.

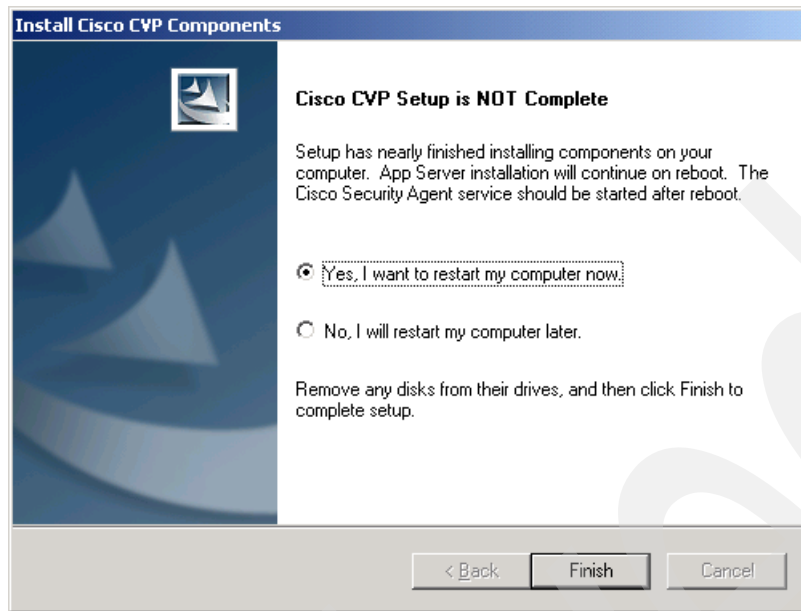


Figure 3-14 Windows Restart Screen

After reboot, the Installer continues, showing several progress screens which require no user intervention. If a screen similar to that shown in Figure 3-13 on page 55 appears, you should again press y and Enter.

After the installer completes, we can continue with the initial configuration procedure, as presented in the next section. It is not necessary to reboot the system again.

### 3.2.2 Initial configuration of the Cisco CVP V3.1 Call Control Server

In this section we perform all the necessary initial configuration tasks to set up the Cisco CVP Call Control Server. In the process, we also take a brief tour around the CVP V3.1 Application Administration screens.

We begin by examining the state of the Cisco CVP Node Manager. As mentioned above, the Cisco CVP Node Manager is responsible for starting, stopping, and restarting the Cisco CVP processes. We can view the current state of these processes using the ICM Service Control facility.

1. Press **Start** → **Programs** → **Cisco Internet Service Node** → **Service Control**

The following dialog box in Figure 3-15 on page 56 opens.

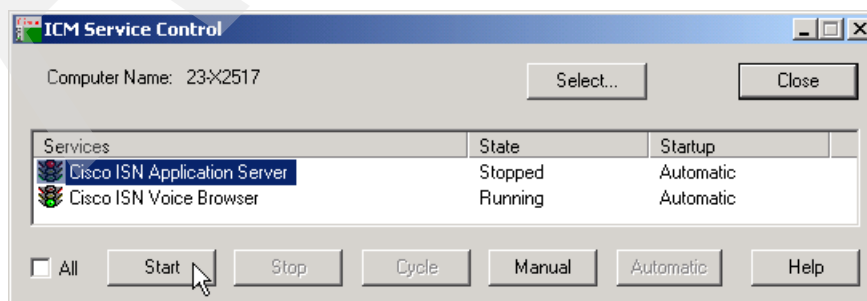


Figure 3-15 ICM Service Control window, immediately following installation

2. Notice that the Cisco CVP Application Server process is stopped, but the Cisco CVP Voice Browser process is running. Start the Cisco CVP Application Server by selecting it and then clicking **Start**.

**Note:** The Cisco CVP Service Control program is very similar to the Windows Service Control facility. Like Windows Service Control, you can set these processes to start automatically upon system startup, or not, and you can start, stop, or cycle (stop and restart) each process individually or as a group. In fact, if you click **All**, the program will show you *all* Windows processes, and the program becomes virtually identical to Windows Service Control.

3. When you see the Cisco CVP Application Server *Running*, click **Close** to close the dialog box.
4. The Cisco CVP Application Administration is accessed with a Web browser interface. It can be reached from any Web browser on the network at the following URL:  
`http://hostname/appadmin`
5. Because we are working directly on the Cisco CVP Call Control Server machine, we use a different technique. Navigate to **Start** → **Programs** → **Cisco Internet Service Node** → **Application Server** → **Application Administrator**. This command brings up the Web page shown in Figure 3-16 on page 57.

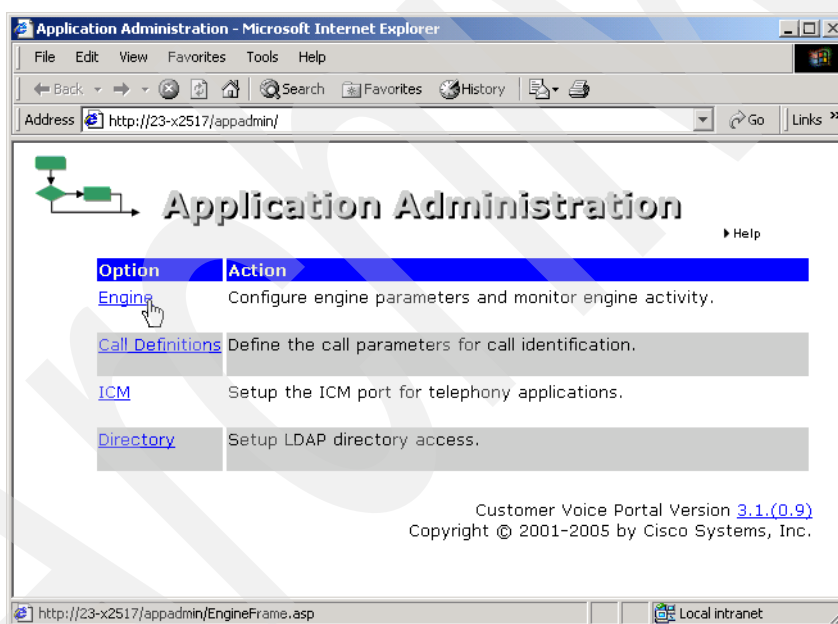


Figure 3-16 Top Level Application Administration screen  
Screen shot reprinted by permission from Microsoft Corporation

6. When you click **Engine**, the Engine Status page appears, as in Figure 3-17 on page 58. Note the highlighted information: the ICM Subsystem is OFFLINE. This means that ICM has not connected to it yet, which is to be expected since we have not yet configured the ICM to do so.

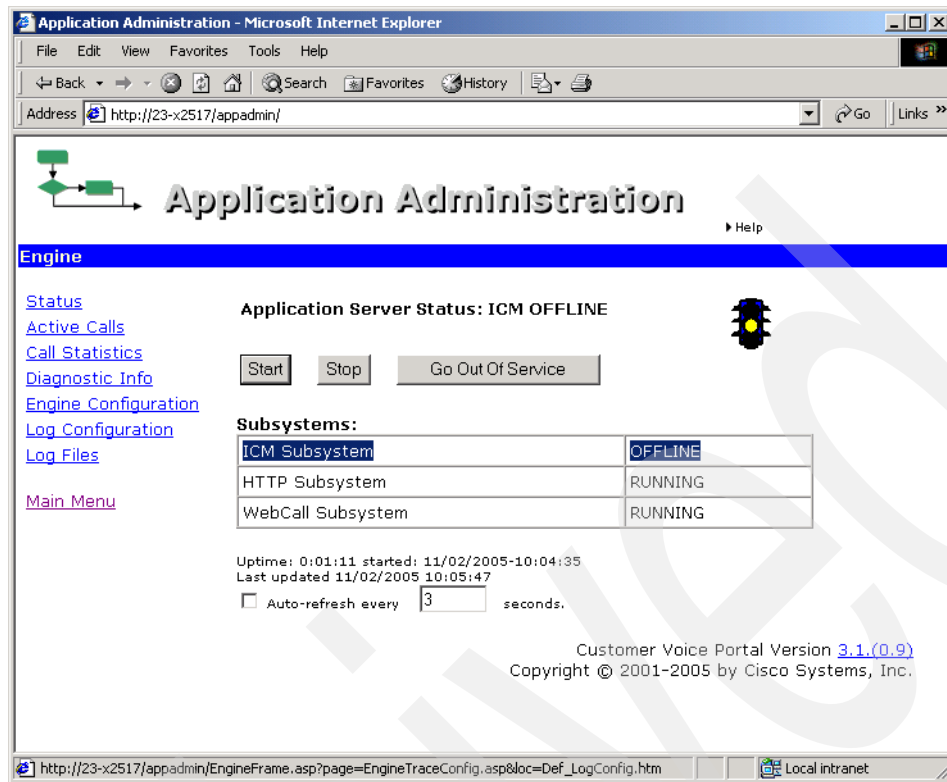


Figure 3-17 Status screen - CVP Application Server is currently offline

Screen capture reprinted by permission from Microsoft Corporation

7. Now click **Engine Configuration**. There are a number of useful options here (see Figure 3-18 on page 59), but there is only one that we need to change at this point. Select **Allow External VXML** and then click **Update**.

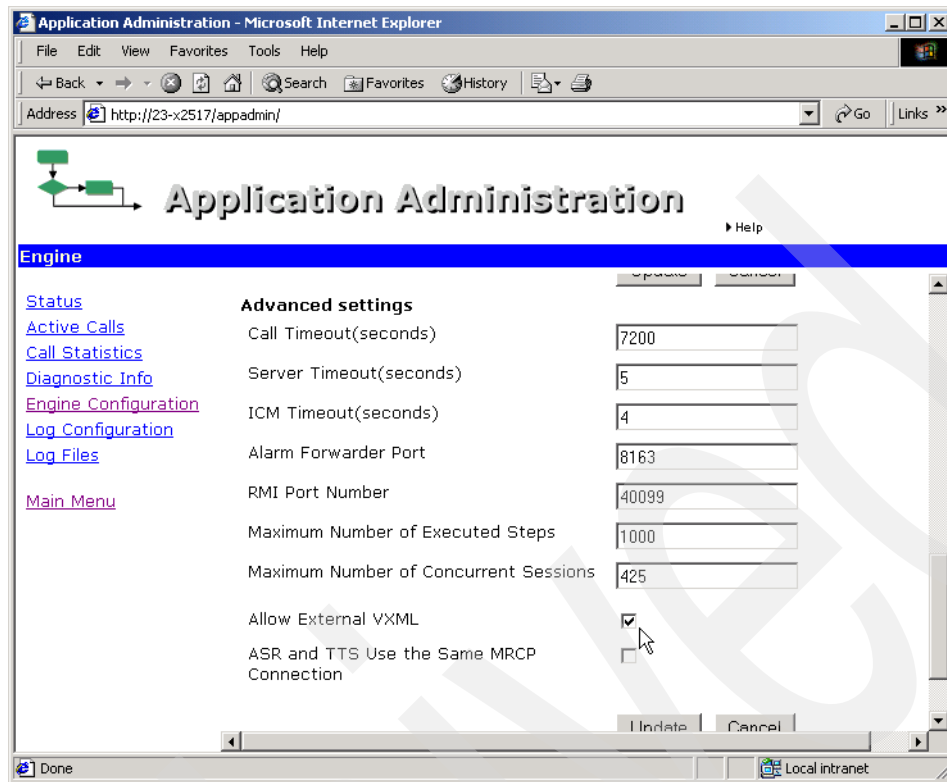


Figure 3-18 Engine Configuration screen - Turn on Allow External VXML

Screen capture reprinted by permission from Microsoft Corporation

8. Now, we turn on a reasonable level of trace logging. Click **Log Configuration** to reach the page shown in Figure 3-19 on page 60. Select **Call** and **Basic** tracing. Leave everything else off, and again remember to click **Update**.

In a production environment, all three of these options will normally be turned off. Then only basic informational, statistical, and error messages will appear in the Cisco CVP Application Server logs. Conversely, for very detailed tracing, you can select **Detailed** tracing, and unselect **Call** and **Basic** tracing. This causes the actual VoiceXML that the Cisco CVP Application Server generates to appear in its logs. Again, always remember to click **Update**.

If you scroll down on the page you can see a long list of individual trace settings. These are never used outside of Cisco's engineering team.

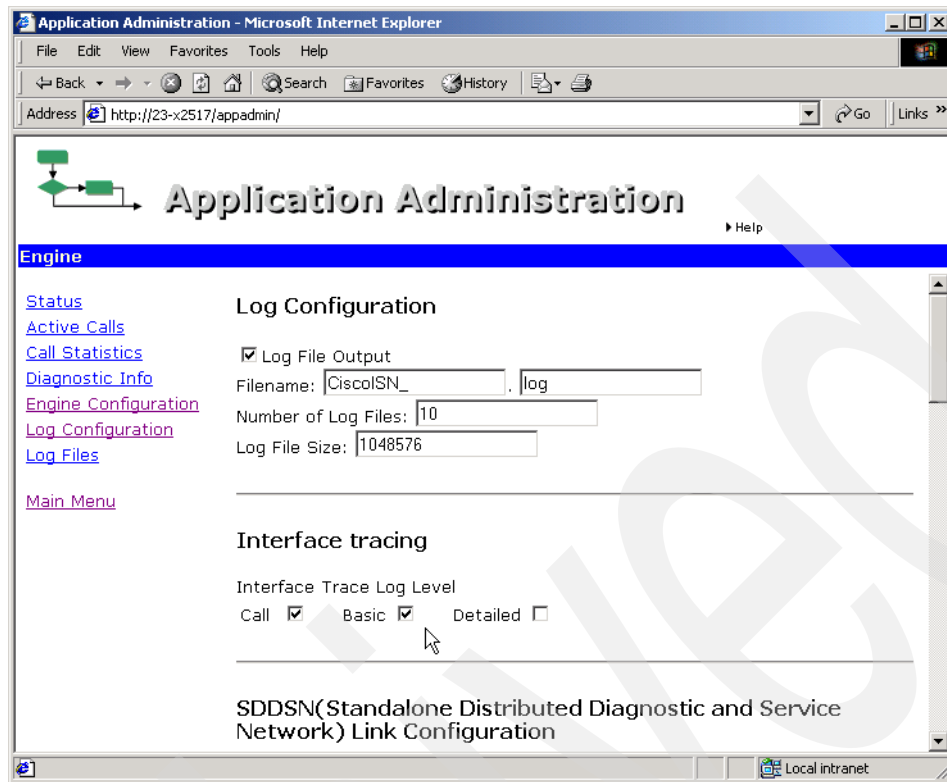


Figure 3-19 Log Configuration screen - Turn on Call and Basic

Screen capture reprinted by permission from Microsoft Corporation

9. Figure 3-20 on page 61 shows where in the directory structure you can find the Cisco CVP Application Server log files. They are stored in plain text and you can open them using Notepad.

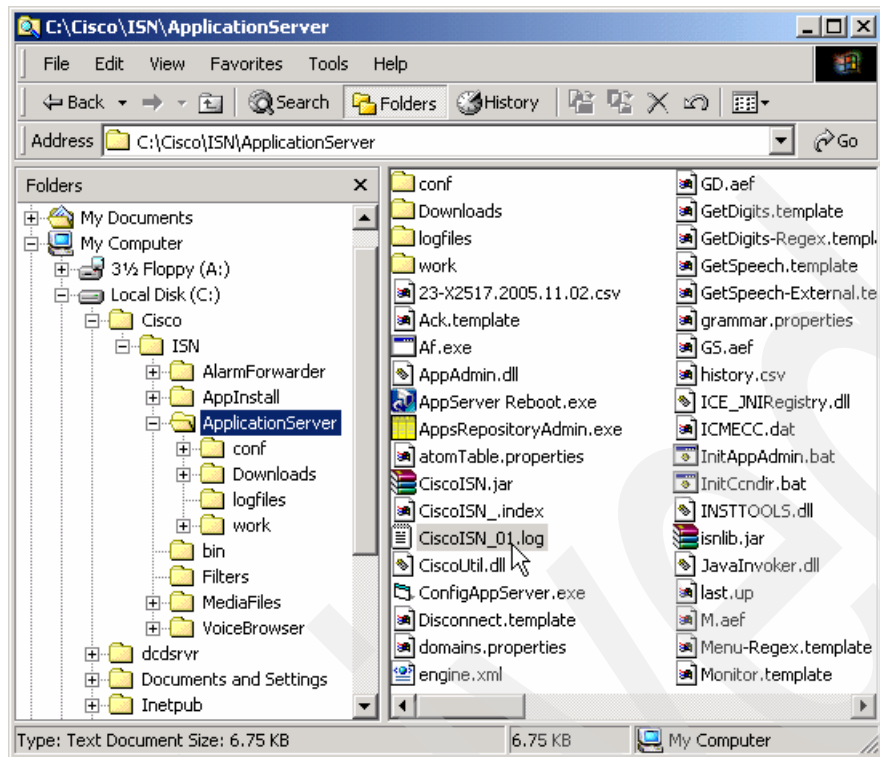


Figure 3-20 Location of CVP Application Server log files

Screen capture reprinted by permission from Microsoft Corporation

10. If you do not happen to be working on the Cisco CVP Call Control Server system directly, you can access these logs from the CVP App Admin Web browser interface. Click **Logs** to reach the page shown in Figure 3-21 on page 61. There you will see a list of log files (this figure shows only one), which you can click to open. Note that depending on what software you have installed on the machine running your web browser, you might have to configure your Web browser so that .asp files are opened using Notepad.

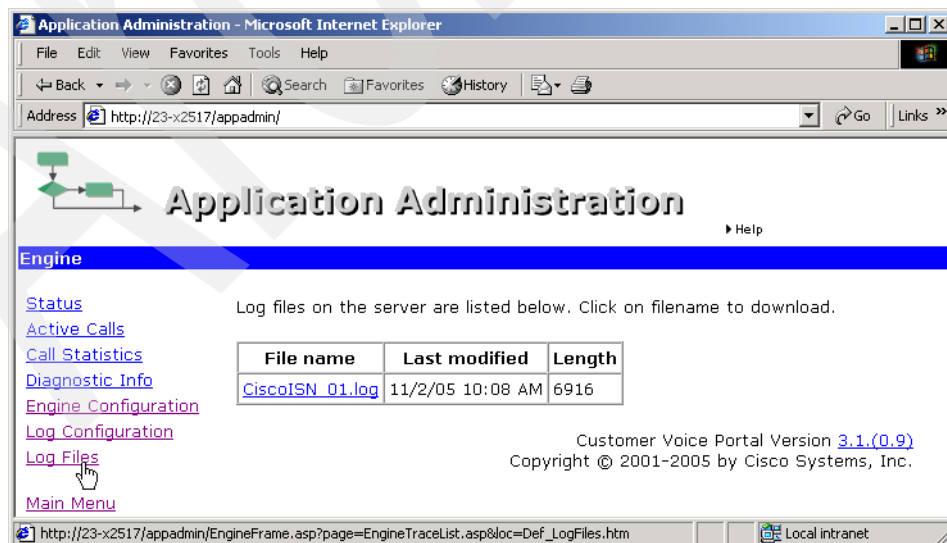


Figure 3-21 Log Files screen - to access log files from a web browser

Screen capture reprinted by permission from Microsoft Corporation

11. Now we can look at the Call Statistics page. Click **Call Statistics** as shown in Figure 3-22.

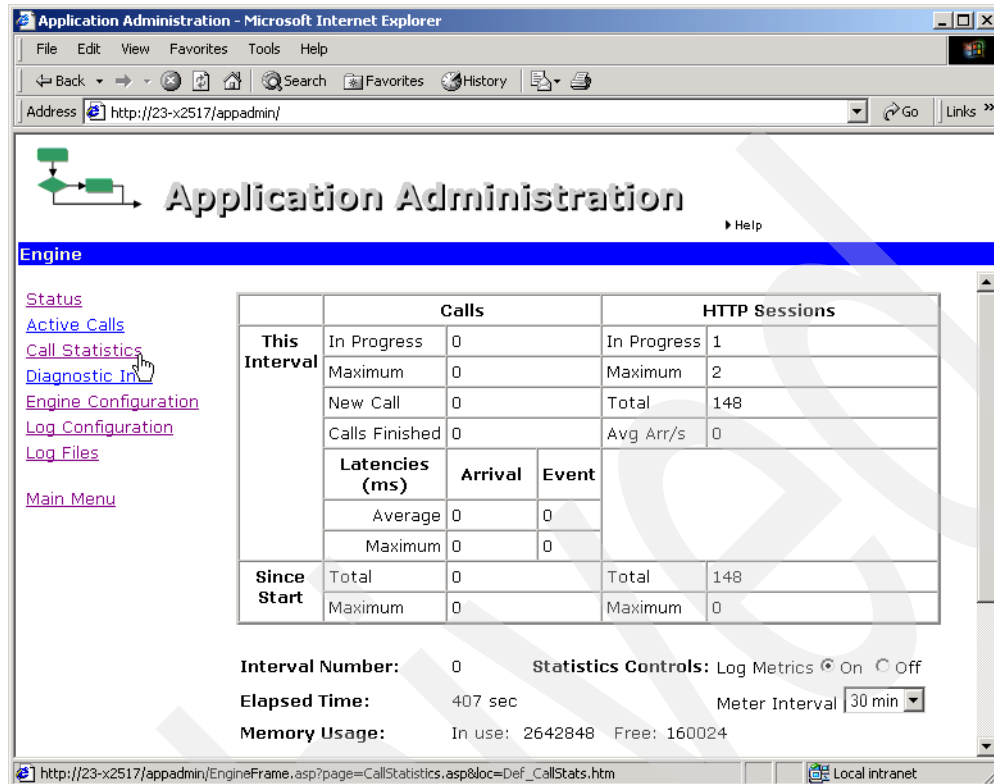


Figure 3-22 CVP Application Server Call Statistics screen

Screen capture reprinted by permission from Microsoft Corporation

The left column shows how many calls this Cisco CVP Call Control Server has taken since the beginning of the current 30-minute interval as well as how many since the process started. The number of calls is somewhat misleading, however. The way calls flow around an ICM Integrated network, some calls end up getting counted twice. Still, you can get an idea of the relative call volume from one period to another.

The lower right corner, contains a menu which allows you to alter the reporting interval. Also, in the lower left corner (not shown in this figure) you can set the page to auto-refresh every few seconds.

The information on this page actually is written automatically to the Cisco CVP Application Server log file at the end of every interval.

12. Click **Main Menu** and then **Call Definitions**. The screen shown in Figure 3-23 opens.



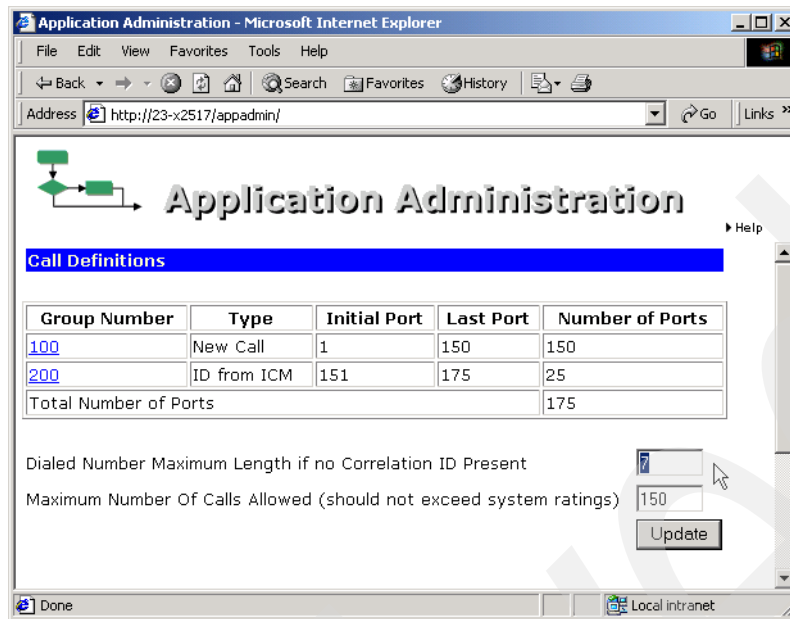


Figure 3-23 Call Definitions screen

Screen capture reprinted by permission from Microsoft Corporation

This screen shows how many calls will be allowed in the system. There are two call groups, 100 and 200. Which kinds of calls use which call group is dependent on how your call network is configured, but in most cases *all* calls go through group 100, and calls which use the VoiceXML Gateway go through group 200 as well. (This is the reason that some calls get double-counted in the Call Statistics page.)

The Cisco CVP Call Control Server is very smart about managing its own call load. The field labeled Maximum Number of Calls Allowed refers to the total of all calls in groups 100 and 200. By default, if the system reaches 99% of that number, it automatically stops accepting new calls. When the load drops to 97%, it begins accepting new calls again. While it is not accepting new calls, the other components in the CVP solution network automatically route calls to other available CVP Call Control Servers.

For our purposes, we leave all of these settings as they are.

Notice the field labeled Dialed Number Maximum Length if no Correlation ID Present. This defaults to 10, but for our setup we need to change it to 7, as shown in the figure. The explanation for this takes us into a discussion of dial plans.

The *dial plan* is the set of rules by which you organize the dialed numbers in your organization. In our lab environment, we have three categories of dialed numbers:

- ▶ Any 4-digit number refers to a VoIP phone destination. You will see that all the dialed numbers used to access the Cisco CVP Standalone applications from IP phones are four digits long.
- ▶ Any 7-digit number beginning with 777 refers to a Plain Old Telephone Service (POTS) call. (In Cisco language, a POTS dial-peer is a dial-peer that can only be matched by T1/T3, E1/E3, or analog calls. It is essentially the opposite of VoIP.) We have used numbers such as 777abcd to provide the POTS equivalents for VoIP numbers abcd.
- ▶ The specific number 8887878 is used by the ICM integration to transfer an incoming call from an Ingress Gateway to a VoiceXML Gateway for VoiceXML processing. In a Cisco CVP configuration, this dialed number is known as the *Voice Response Unit (VRU) Routing Number*.

The last bullet is actually the reason why we need to set the field labeled Dialed Number Maximum Length if no Correlation ID Present to 7. It must be set to the number of digits in the VRU Routing Number.

1. After setting that value, click **Update**. If you receive the message shown in Figure 3-24, you may safely click **OK**.

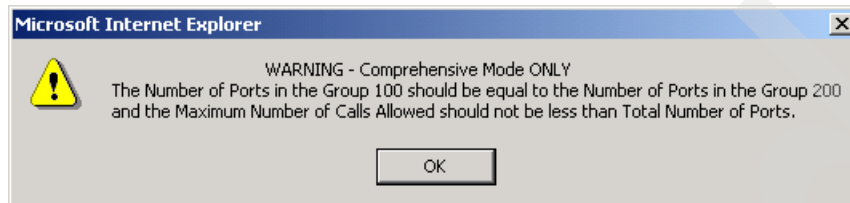


Figure 3-24 Warning message regarding the Number of Ports

2. Now, we look at the ICM Subsystem Configuration page, which is shown in Figure 3-25 on page 64. Click **Main Menu** and then **ICM**.

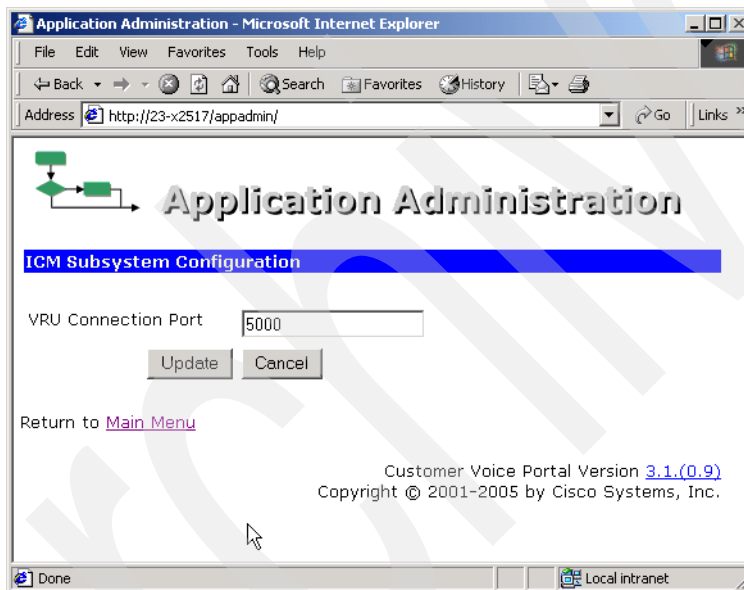


Figure 3-25 ICM Subsystem Configuration screen

Screen capture reprinted by permission from Microsoft Corporation

3. In the world of TCP/IP, ICM is the *client*, and the Cisco CVP Application Server is the *server*. This means that ICM connects to Cisco CVP, rather than the other way around. All we need to do on the server side is to configure the TCP/IP port number to which ICM will connect. This value defaults to 5000, and we leave it as such. We will see in **Section TBD** where we configure the ICM to connect to this port.

This completes our exploration of the Cisco CVP Application Server. The other major component in the Cisco CVP Call Control Server is the Cisco CVP Voice Browser.

1. Navigate to **Start → Programs → Cisco Internet Service Node → Voice Browser → VB Admin**. A screen similar to the one shown in Figure 3-26 on page 65 opens.

```
***** ISN Voice Browser Configuration Utility *****
>>>>showgatekeeper
No Gatekeeper is currently set
>>>>setgatekeeper /?
IP address for the Gatekeeper serving the Voice Browser. A value of none means
the gatekeeper is not
being used. Calls cannot be transferred without a gatekeeper.
Valid Values: IP Address or none
Default: none
System shutdown and startup necessary for changes to take effect.
ShowGatekeeper [/? ! /help]
SetGatekeeper [<New Value> ! /? ! /help]
>>>>setgatekeeper 9.42.171.186
Gatekeeper IP Address has been changed from "none" to "9.42.171.186"
The new setting will be updated upon system startup
>>>>showtechprefix
The Tech Prefix is "2#"
>>>>settechprefix 1#
The Tech Prefix has been changed from "2#" to "1#"
The new setting will be updated upon system startup
>>>>showappserverlist
The list of Application Servers is "localhost:8000/servlet/ism "
>>>>quit_
```

Figure 3-26 CVP Voice Browser Administration (VB Admin) settings

VB Admin is a command line utility which allows all of the Cisco CVP Voice Browser settings to be examined and modified. Generally, each setting has a Show version and a Set version. You can see the full list of commands by typing `mhlp` - or by reading the Cisco documentation.

For initial configuration purposes, we need to set the following parameters:

- IP address of the Gatekeeper

This tells the Gatekeeper that incoming calls from the Ingress Gateway may be sent to this Cisco CVP Voice Browser

- Tech Prefix

Technology Prefix is a technique used to *partition* the dialed number plan in a Gatekeeper-controlled network. For our purposes we will always use Tech Prefix 1#

- CVP Application Server list

This provides the Cisco CVP Voice Browser with a list of Cisco CVP Application Servers to which it can forward calls, on their way to ICM. Normally one would point only to the Cisco CVP Application Server that resides on the same Cisco CVP Call Control Server, which is the default configuration.

These settings are shown in Figure 3-26 on page 65. To exit the VB Admin utility, you must type `quit`, not `exit`.

2. Note that some of the settings require a restart in order to take effect. To restart the Cisco CVP Voice Browser, navigate to **Start** → **Programs** → **Cisco Internet Service Node** → **Service Control**. Select **Cisco ISN Voice Browser**, and click **Cycle** as shown in Figure 3-27 on page 66.

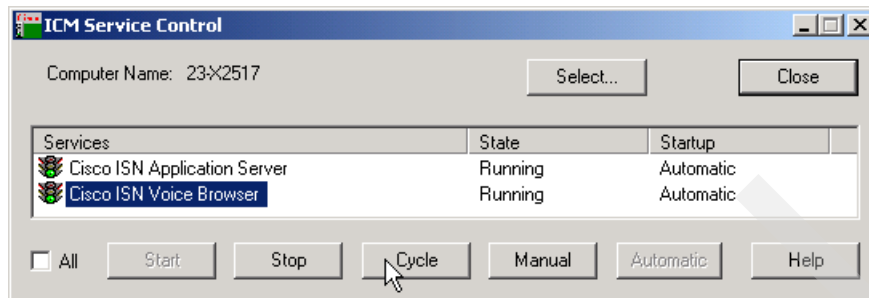


Figure 3-27 Restarting the CVP Voice Browser

We are now ready to configure the Gateway and Gatekeeper.

### 3.2.3 Initial Gateway and Gatekeeper Configuration for ICM Integrated Deployments

In this section, we prepare both the Gateway and the Gatekeeper for use with ICM Integrated environments. As before, we discuss in this section only those items which are global. We do not cover configuration items that are specific to a particular test or application.

We begin by downloading a series of Cisco CVP-provided files into the Gateway's flash memory. These files are different from the ones used in Cisco CVP Standalone mode.

We recommend setting up a TFTP server. There are several free TFTP server utilities available on the internet. One such free utility can be downloaded from:

[http://www.solarwinds.net/Tools/Free\\_tools/TFTP\\_Server/index.htm](http://www.solarwinds.net/Tools/Free_tools/TFTP_Server/index.htm)

Figure 3-28 on page 67 shows the area on the disk where the Cisco CVP Application Server leaves all the downloadable gateway files. There are actually *eight* sets of files, each with slight differences, depending on whether you want automatic call restart after failure (we do not), whether you are using a Cisco Content Services Switch, (we are not), and whether you are using the CVP Voice Browser to accept incoming calls (we are). All the files shown in the appropriate directory must be downloaded to the gateway.

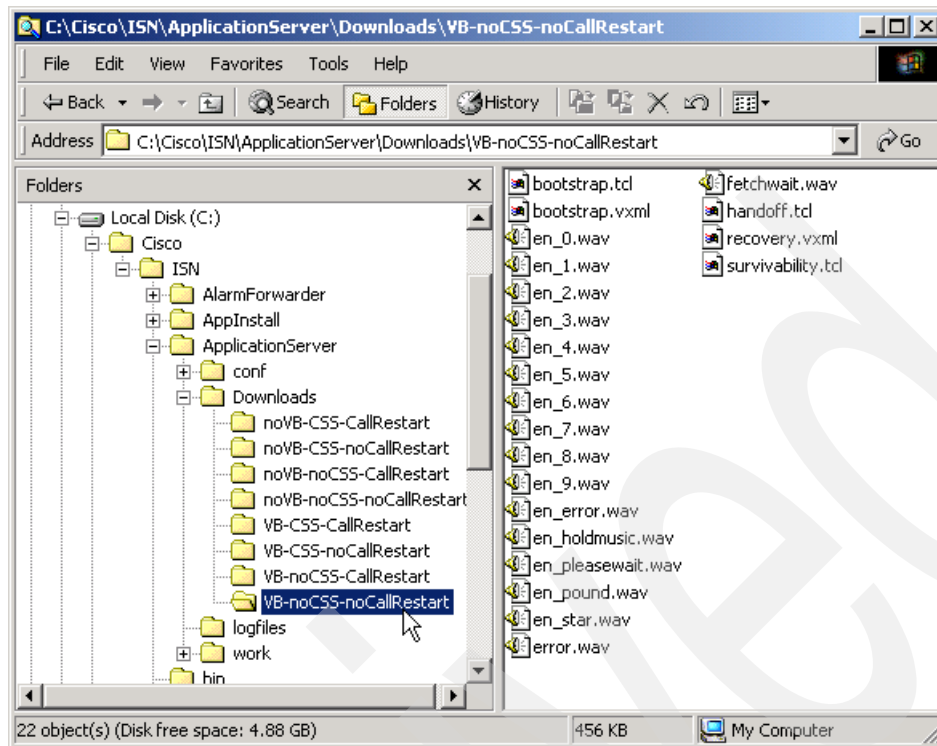


Figure 3-28 Files which must be downloaded to the gateway

Screen shot reprinted by permission from Microsoft Corporation

In Figure 3-29 on page 67, we see some of the gateway commands for downloading files from a TFTP server.

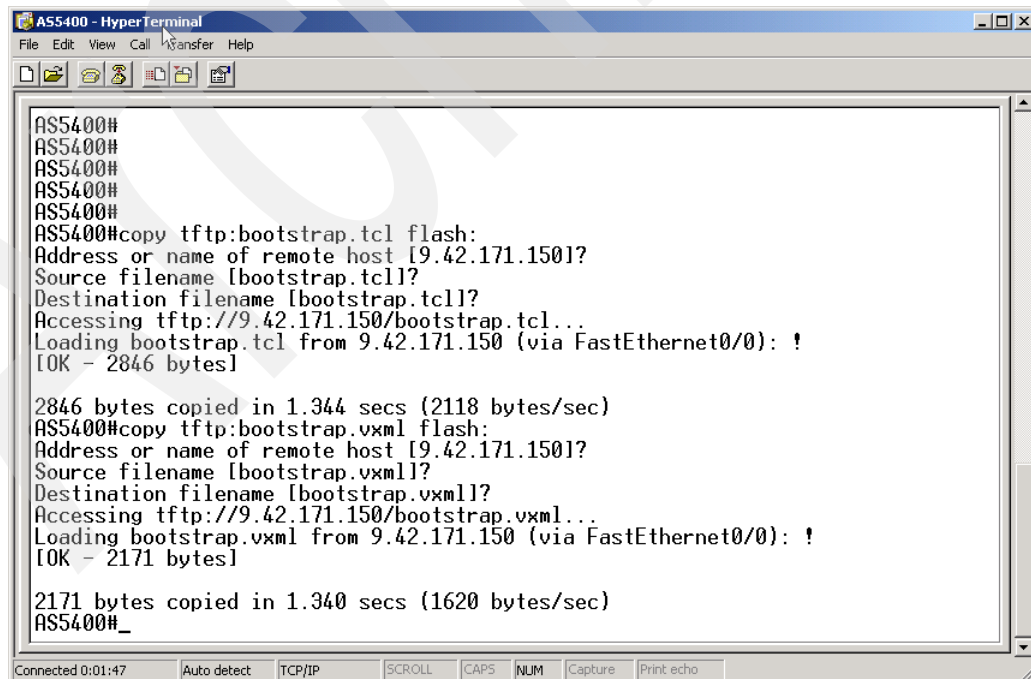


Figure 3-29 Download CVP files to flash memory

Screen shot reprinted by permission from Microsoft Corporation

When the downloads are completed, a directory of the flash memory displays the files as shown in Figure 3-30.

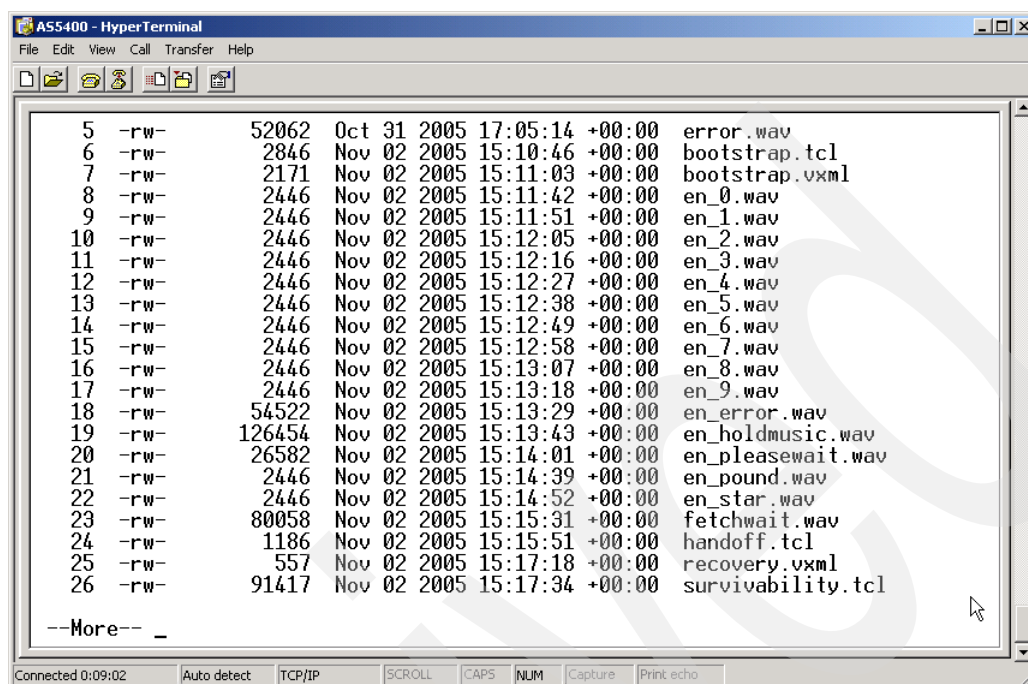


Figure 3-30 Flash Memory directory listing  
Screen shot reprinted by permission from Microsoft Corporation

Now, we need to perform a series of configuration functions at the gateway. Much of the necessary configuration was already performed in “Configure the Cisco Gateway for Cisco CVP Standalone Mode” on page 33. In this section, we only discuss items that must be configured above and beyond what we have already done.

In the discussion that follows, we present excerpts from the Cisco AS5400HPX Universal Gateway and the Cisco 3660 Gatekeeper configurations. The full running-configurations can be found in Appendix A, “Cisco Router Configurations” on page 347.

**Note:** Configuration lines might not appear here in the same order they appear in the running-config.

Earlier, we configured some lines in the FastEthernet section. Here, we add the configuration lines show in Example 3-1.

*Example 3-1 Configuration lines to add to the FastEthernet section*

```
interface FastEthernet0/0
...
h323-gateway voip interface
h323-gateway voip id Gatekeeper-3660 ipaddr 9.42.171.186 1719
h323-gateway voip h323-id 9.42.171.128
h323-gateway voip tech-prefix 1#
```

Remember that the ICM Integrated deployment requires a Gatekeeper, while the Standalone deployment does not. These lines are necessary now to register the Gateway with the Gatekeeper.

In our setup, 9.42.171.186 is the address of the 3660 Gatekeeper, and 9.42.171.128 is the address of the Cisco AS5400HPX Universal Gateway (which we are configuring now). Also, notice that here is where we set the Technology Prefix for this Gateway to 1#.

The lines shown in Example 3-2 are equivalent to lines in an /etc/hosts file in Unix or Windows. They simply provide for local mapping of those names to the corresponding IP addresses. The particular names shown in Example 3-1 are used only by CVP Microapplications, operations such as Play Media, Get and Play Data, Get Speech. These operations are invoked directly by the ICM and do not use the CVP VoiceXML Server. One can actually build an entire IVR application this way, but it is very painful to do so. It is much easier to use the CVP VoiceXML Studio and Server for that purpose.

---

*Example 3-2 Local mapping of names to IP addresses*

---

```
ip host asr-en-us 9.42.171.89
ip host asr-en-us-backup 9.42.171.89
ip host tts-en-us 9.42.171.89
ip host tts-en-us-backup 9.42.171.89
ip host mediaserver 9.42.171.24
ip host mediaserver-backup 9.42.171.24
```

---

On the other hand, most customers who use ICM Integrated deployments will use the Microapplications to a certain degree. They are useful for prompt and collect activities prior to transferring into a true self-service application or to a contact center agent. They are also useful for playing music files and periodic voice messages while a caller is waiting in queue. VoiceXML Server ports are much too expensive to use for such lightweight purposes.

Our Redpaper applications do not actually invoke any Cisco CVP Microapplications (except the special one needed to pass control to a VoiceXML Server application itself). The previous commands are therefore extraneous for us, but we include them here for illustrative purposes.

The two host mappings show in Example 3-3 *are* required. They are necessary to point incoming calls to the Cisco CVP Call Control Server. If you are using a Cisco Content Services Switch, you would point these both to the same virtual IP (VIP) address. Otherwise, you could point one to a primary Cisco CVP Call Control Server and one to a backup.

---

*Example 3-3 Required host mappings*

---

```
ip host isn-vxml 9.42.171.150
ip host isn-vxml-backup 9.42.171.150
```

---

The Application Service definitions shown in Example 3-4 are universal. They do not need to be adapted for each application. They are required in order to start voice browser sessions in the VoiceXML Gateway when calls are transferred by ICM to the VRU side.

---

*Example 3-4 Application Service definitions*

---

```
application
service new-call flash:bootstrap.vxml
  paramspace english language en
  paramspace english index 0
  paramspace english location flash:
  paramspace english prefix en
!
service handoff flash:handoff.tcl
  paramspace english language en
  paramspace english index 0
  paramspace english location flash:
```

```

    paramspace english prefix en
!
service bootstrap flash:bootstrap.tcl
    paramspace english language en
    paramspace english index 0
    paramspace english location flash:
    paramspace english prefix en
!

```

---

**Note:** Whenever you enter or modify entries in the Application Service section of the configuration, you must instruct the gateway to reload the application. This is done after exiting from configuration mode, by entering the command **call application voice load name**, where *name* represents each Application Service you have modified. Applications also must be loaded if you modify the flash memory files to which they point.

The two dial-peers shown in Example 3-5 are necessary to handle incoming POTS calls for ICM. (For ICM Integrated applications, we can only use POTS phones to make incoming calls. Using IP phones is possible, but not with the capabilities provided by the freeware H.323 softphone we are using in the Redpaper lab.)

#### *Example 3-5 Dial-peers*

```

dial-peer voice 7774 pots
    incoming called-number 777....
    direct-inward-dial
!
dial-peer voice 77740 voip
    translation-profile incoming block
    destination-pattern 777....
    session target ras
    tech-prefix 1#
    dtmf-relay rtp-nte h245-signal h245-alphanumeric
    codec g711ulaw
    no vad
!

```

---

Our incoming calls will be seven digits long, and always begin with 777. Such a call, when invoked by a POTS phone, will select the POTS dial-peer named 7774 shown here. Since dial-peer 7774 contains no explicit routing instructions, the gateway looks to send the incoming POTS call out again on an appropriate VoIP dial-peer. The dialed number locates the VoIP dial-peer labeled 77740. In this dial-peer, the line session target *ras* tells the gateway to ask the gatekeeper what IP address the dialed number corresponds to.

Example 3-6 shows the dial-peer to which ICM transfers the call in order to start a VoiceXML session. Notice the incoming called-number, 8887878T. This is the VRU Routing Number that we chose for transfers to the VoiceXML Gateway. It shows up in the Gatekeeper configuration as well as in the ICM.

#### *Example 3-6 Dial-peer to which ICM transfers the call in order to start a VoiceXML session*

```

dial-peer voice 8887878 voip
    translation-profile incoming block
    service bootstrap
    incoming called-number 8887878T
    dtmf-relay rtp-nte h245-signal h245-alphanumeric
    codec g711ulaw
    no vad

```

---



The suffix T is a wildcard indicator, and is very important. Remember that when ICM transfers the call to the VoiceXML Gateway, it tacks a *Correlation ID* on to the end of the VRU Routing Number. Adding the T makes it possible for this dial-peer to be selected by VRU Routing Number transfers which arrive at this gateway, no matter what Correlation ID has been attached to the dialed number.

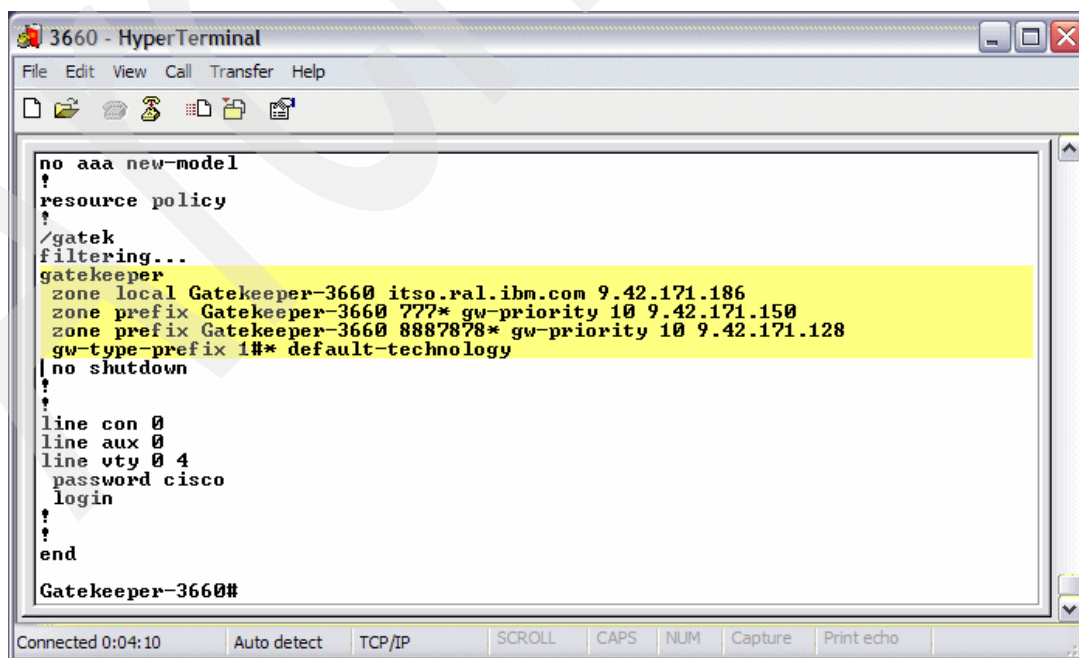
Now, we get to configure the Gatekeeper. As described in, “Cisco Gatekeeper” on page 7. The Gatekeeper is used as a directory server for calls that arrive at an Ingress Gateway, as well as for calls that are being transferred around the VoIP network. Given a dialed number, the Gatekeeper returns the IP address to which that call should next be routed.

**Note:** As part of CVP’s high availability design, the Gatekeeper can load-balance among possible IP addresses, and resort to backup IP addresses if particular configured destinations are not ready to receive calls.

Figure 3-31 on page 71 shows two configuration lines which are enough for the purposes of this Redpaper. Incoming POTS calls to dialed numbers beginning with 777 will be routed to our Cisco CVP Call Control Server at To Be Determined (TBD) address and calls transferred to the VRU Routing Number, 8887878, will be delivered to the AS5400HPX VoiceXML Universal Gateway at 9.41.171.128.

**Note:** The wildcard character that follows the VRU Routing Number is required because of the way ICM transfers calls to the VoiceXML Gateway. It actually tacks on some extra digits at the end of the VRU Routing Number which are known as the *Correlation ID*. When ICM receives notification of a transferred call to the VoiceXML Gateway, it matches the incoming Correlation ID with the outgoing Correlation ID, and thereby determines that these two call legs are actually the same call.

Finally, notice that the Gatekeeper’s default Technology Prefix is set to 1# to match all of our other components.



```
no aaa new-model
?
resource policy
?
/gatek
filtering...
gatekeeper
zone local Gatekeeper-3660 itso.ral.ibm.com 9.42.171.186
zone prefix Gatekeeper-3660 777* gw-priority 10 9.42.171.150
zone prefix Gatekeeper-3660 8887878* gw-priority 10 9.42.171.128
gw-type-prefix 1#* default-technology
?
no shutdown
?
line con 0
line aux 0
line vty 0 4
password cisco
login
?
end
Gatekeeper-3660#
```

Figure 3-31 Gatekeeper Configuration

We have now completed the initial Gateway and Gatekeeper configuration.

### 3.2.4 ICM Configuration

We begin our ICM configuration exercise by assuming that much of it, the part that does not relate to CVP, is already done. In our case, we have installed an ICM Logger, Router, Distributor, and one Peripheral Gateway (PG1A) on our all-in-one ICM machine. These processes can be seen by opening ICM Service Control, as illustrated in Figure 3-32 on page 72.

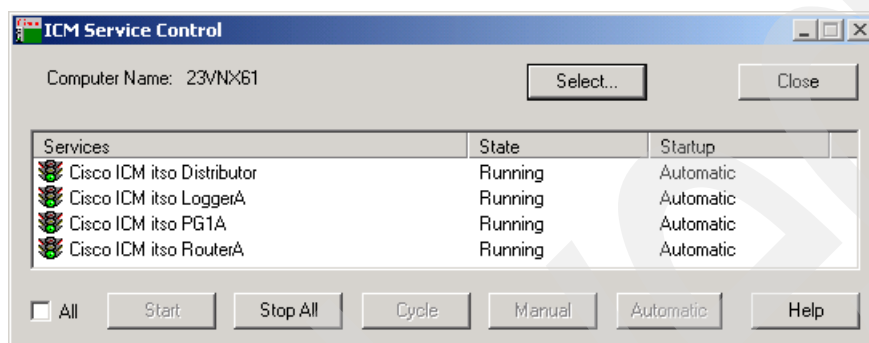


Figure 3-32 ICM Service Control: What's already installed and running

We have not yet added any peripherals to our Peripheral Gateway, and in fact in our lab, Cisco CVP will be the *only* peripheral we add.

There are two parts to adding a peripheral:

1. Configure the peripheral in the ICM Configuration Manager.
  2. Add the appropriate software components using ICM Setup.
1. We begin by starting the ICM Configuration Manager by double-clicking on the appropriate icon in the ICM Admin Workstation program group. This is shown in Figure 3-33. Navigating **Start** → **Programs** → **ICM Admin Workstation** → **Configuration Manager** would also get us there.

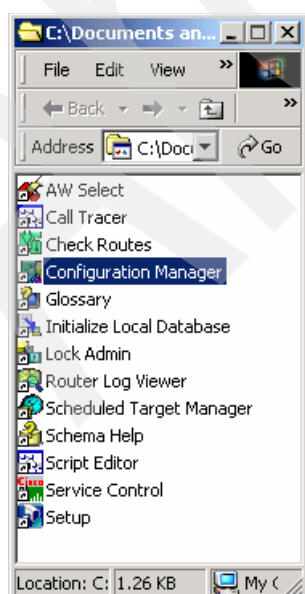


Figure 3-33 Starting the ICM Configuration Manager

The ICM Configuration Manager contains a number of tools. In this setup we use the PG Explorer, the Network VRU Explorer, and the Network VRU Script List tools.

2. We begin by double-clicking **Tools** → **Explorer Tools** → **PG Explorer**, as shown in Figure 3-34.

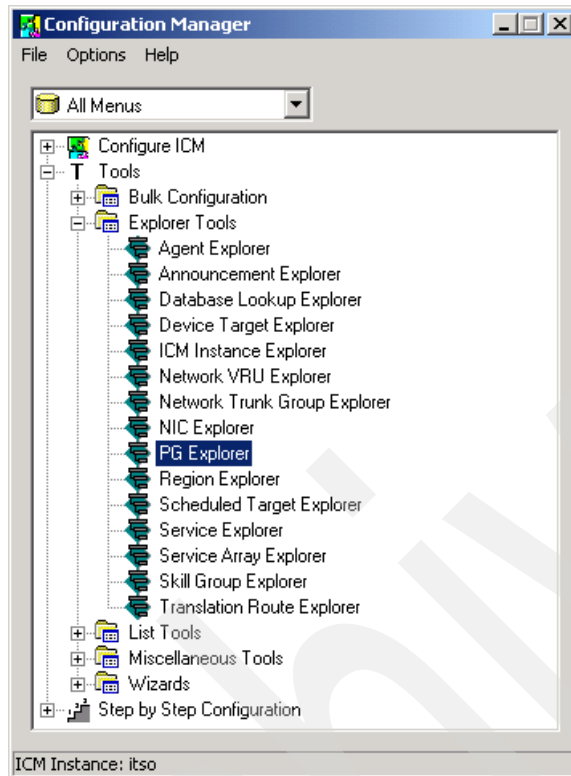


Figure 3-34 Start the PG Explorer tool

3. All ICM Configuration Manager tools have a Retrieve button. Whenever we open a tool we click that button to direct the tool to retrieve all the relevant data from the database. This is shown in Figure 3-35 on page 74.

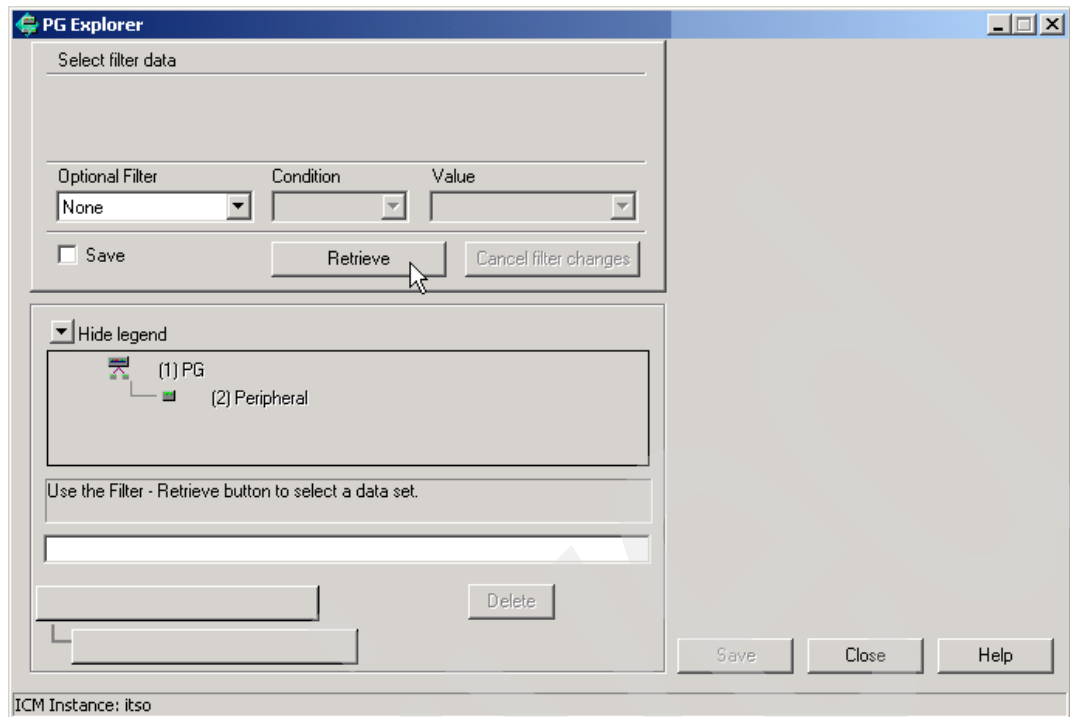


Figure 3-35 Always press Retrieve when opening an ICM Configuration Manager tool.

4. When we are in the PG Explorer, click **Add PG** and then click **Add Peripheral**. This opens up two panes on the right side of the tool window. The top pane refers to the Peripheral Gateway that can control multiple peripherals, and the bottom pane refers to the particular peripheral we are adding.
5. In the top pane, set the Name field value to PG1, and select **VRU** from the “Client type” menu list.
6. In the bottom box, perform the following actions:
  - Set the Name field value to CVP.
  - Set the Peripheral name field value to the *hostname* where the CVP Call Control Server is running.
  - Select **VRU** from the Client type menu list.
  - Select **Enable post-routing**.

These entries are shown in Figure 3-36 on page 75.

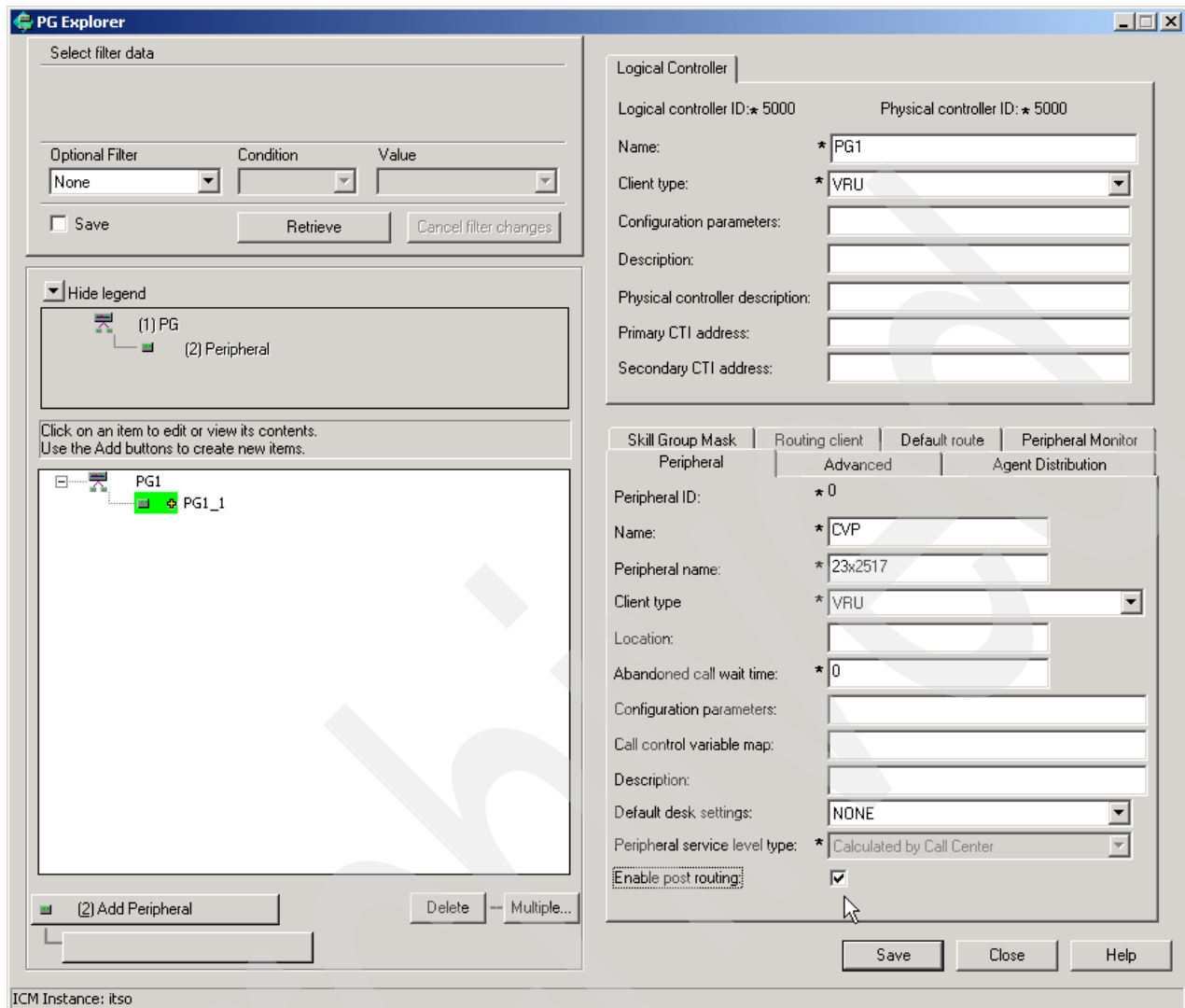


Figure 3-36 PG Explorer, Logical Controller and Peripheral tabs

7. On the Routing Client tab, perform the following actions:

- ▶ Set the Name field value to CVP.
- ▶ Select **Cisco\_Voice** from the Default Media Routing Domain menu list.
- ▶ Select **VRU** from the Client type menu list
- ▶ Select **Network Transfer Preferred**.

These settings are shown in Figure 3-37 on page 76.

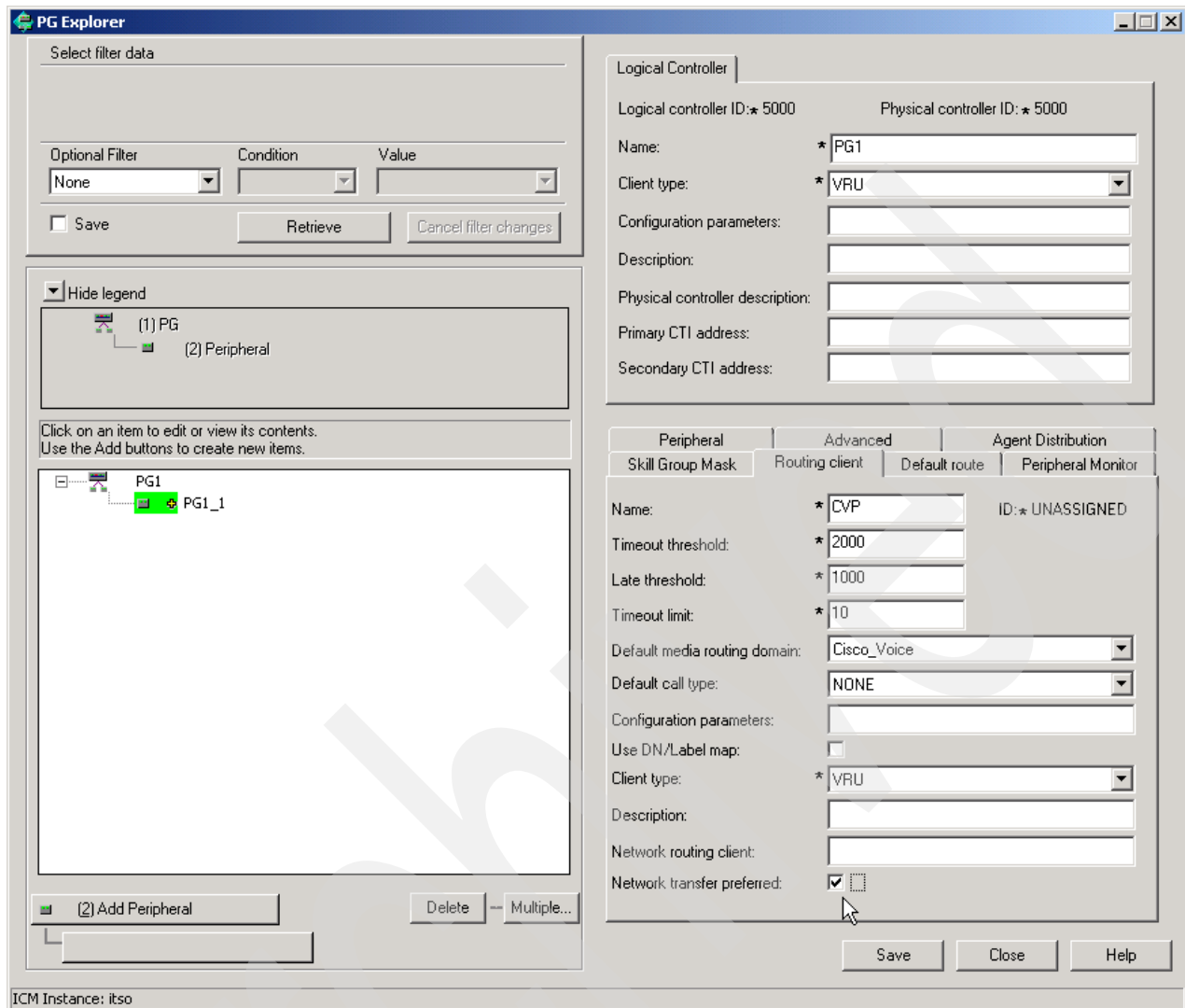


Figure 3-37 PG Explorer, Routing Client tab

8. Click **Save**.

After the tool saves your data, be sure to return to the **Peripheral** tab. You should see something similar to Figure 3-38 on page 77.

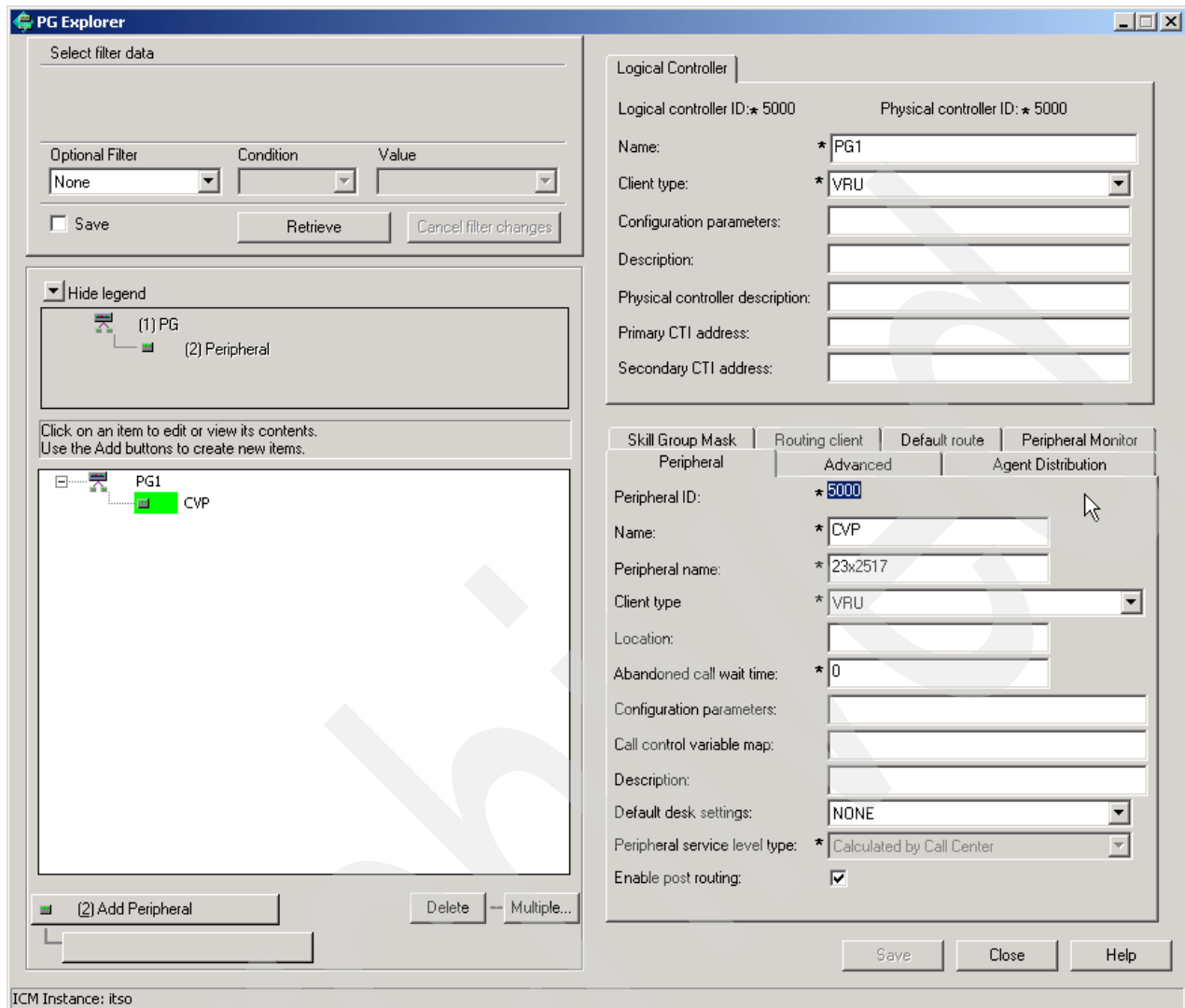


Figure 3-38 PG Explorer, after pressing Save

There are two numbers you must write down at this point for later use.

- ▶ Logical controller ID (on the upper right pane)
- ▶ Peripheral ID (on the lower right pane)

In our case, both of these numbers are 5000, but this will not always be the case. Do not close the PG Explorer tool yet because we will return to it soon.

9. Go back to the ICM Configuration Manager and double-click **Tools** → **Explorer Tools** → **Network VRU Explorer**, as shown in Figure 3-39 on page 78.

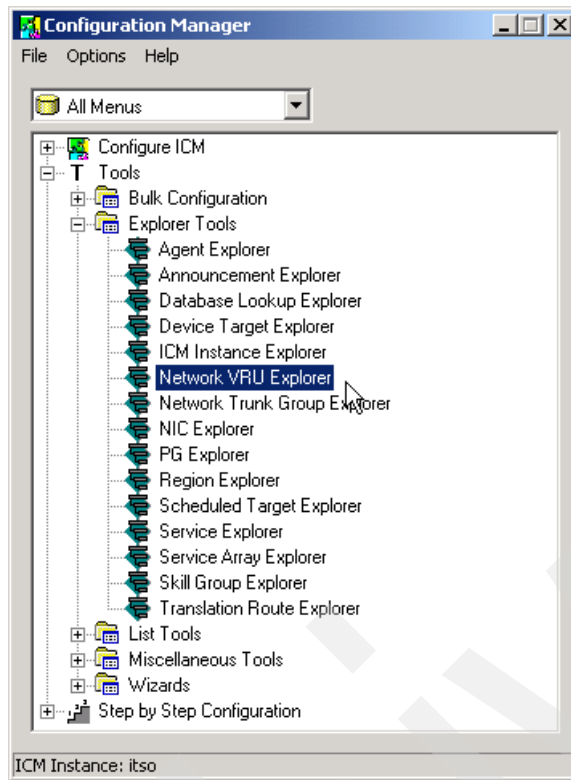


Figure 3-39 Opening the Network VRU Explorer

10. Figure 3-40 on page 78 shows the Network VRU Explorer dialog. Click **Retrieve** to retrieve any existing data from the database.

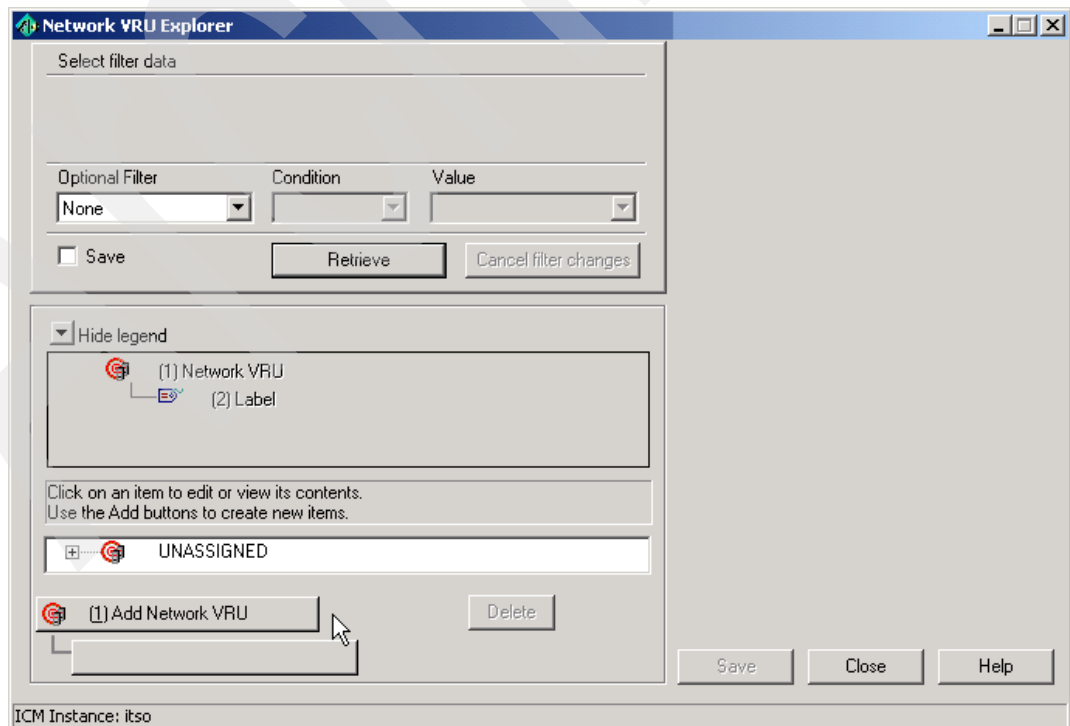


Figure 3-40 Network VRU Explorer tool



For our purposes, we must add two Network VRUs. The first one corresponds to the call Ingress Gateway part of a Cisco CVP call, and the second corresponds to the VoiceXML Gateway part of a Cisco CVP call. The first one is responsible for call switching, so we will call it *CVPSwitch*. The second one which performs VoiceXML activity. We will call it *CVPVRU*. We then associate CVPSwitch to the Cisco CVP peripheral, and we specify in the CVPVRU configuration what dialed number (or *label*, in ICM terminology) transfers the call from CVPSwitch to CVPVRU.

11. Click **Add Network VRU** and then **Add Label**.

12. In the upper pane, set the Name field value to CVPSwitch, and select **Type 5** from the Type menu list.

13. In the lower pane, select **CVP** from the Routing Client menu list, and set the Label field value to DUMMY.

These settings are shown in Figure 3-41 on page 79.

14. Click **Save**.

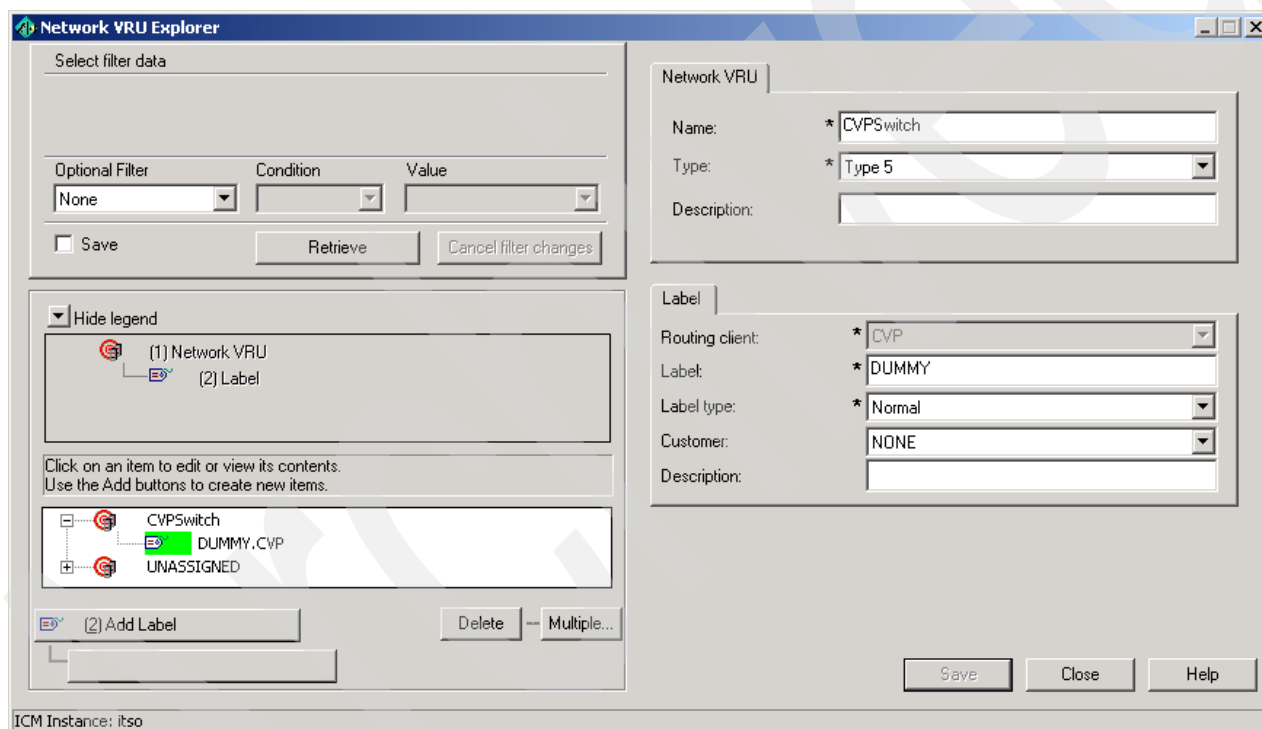


Figure 3-41 Adding the CVPSwitch Network VRU

15. To add the CVPVRU Network VRU, click **UNASSIGNED**, click **Add Network VRU**, and then click **Add Label**.

- In the upper pane, set the Name field value to CVPVRU, and select **Type 7** from the menu for the Type field.

- In the lower pane, select Routing Client from the menu for the Routing Client field and set the Label field value to 8887878. You may recognize this as the VRU Routing Number, the dialed number used to transfer the call to the VoiceXML Gateway.

These settings are shown in Figure 3-42 on page 80.

16. Click **Save**.

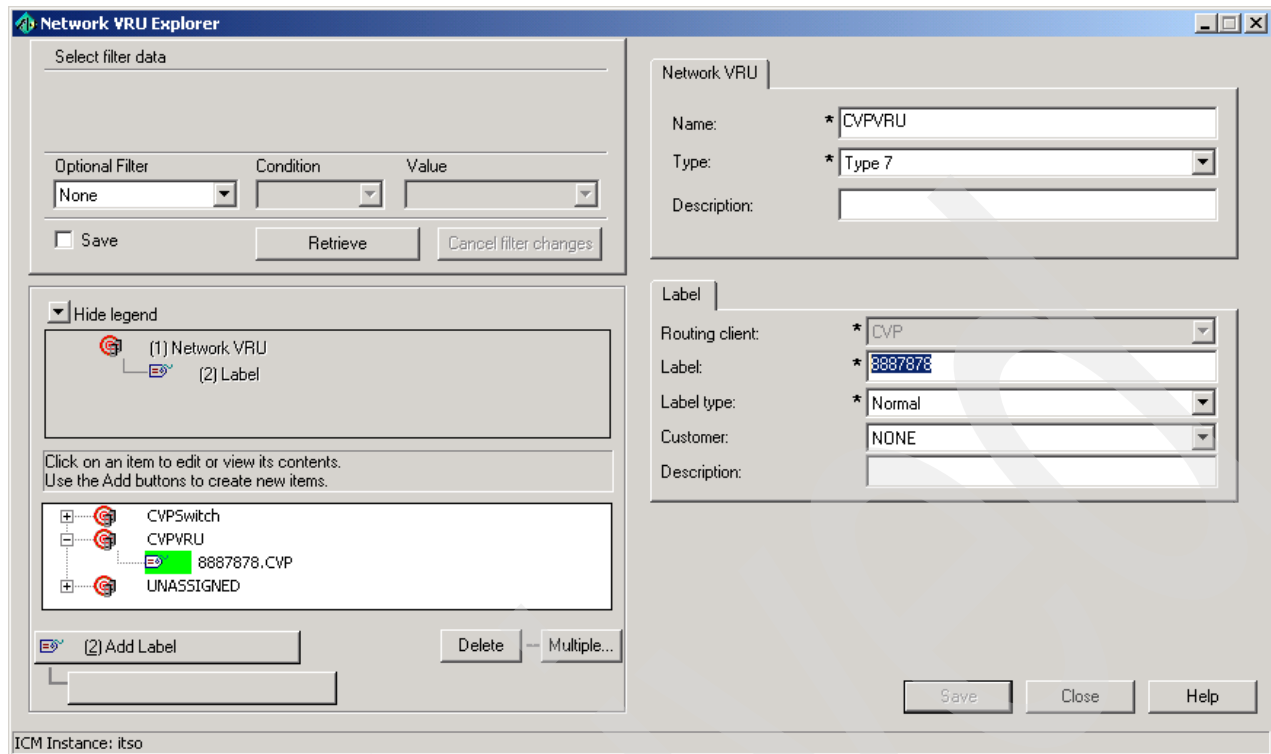


Figure 3-42 Adding the CVPVRU Network VRU

Now that we have added the necessary Network VRUs, we need to go back to the PG Explorer and associate the CVPSwitch Network VRU to the Cisco CVP peripheral.

17. In the PG Explorer dialog box, click **Advanced** and select **CVPSwitch** from the Network VRU menu list, as shown in Figure 3-43. Click **Save** and then click **Exit**.

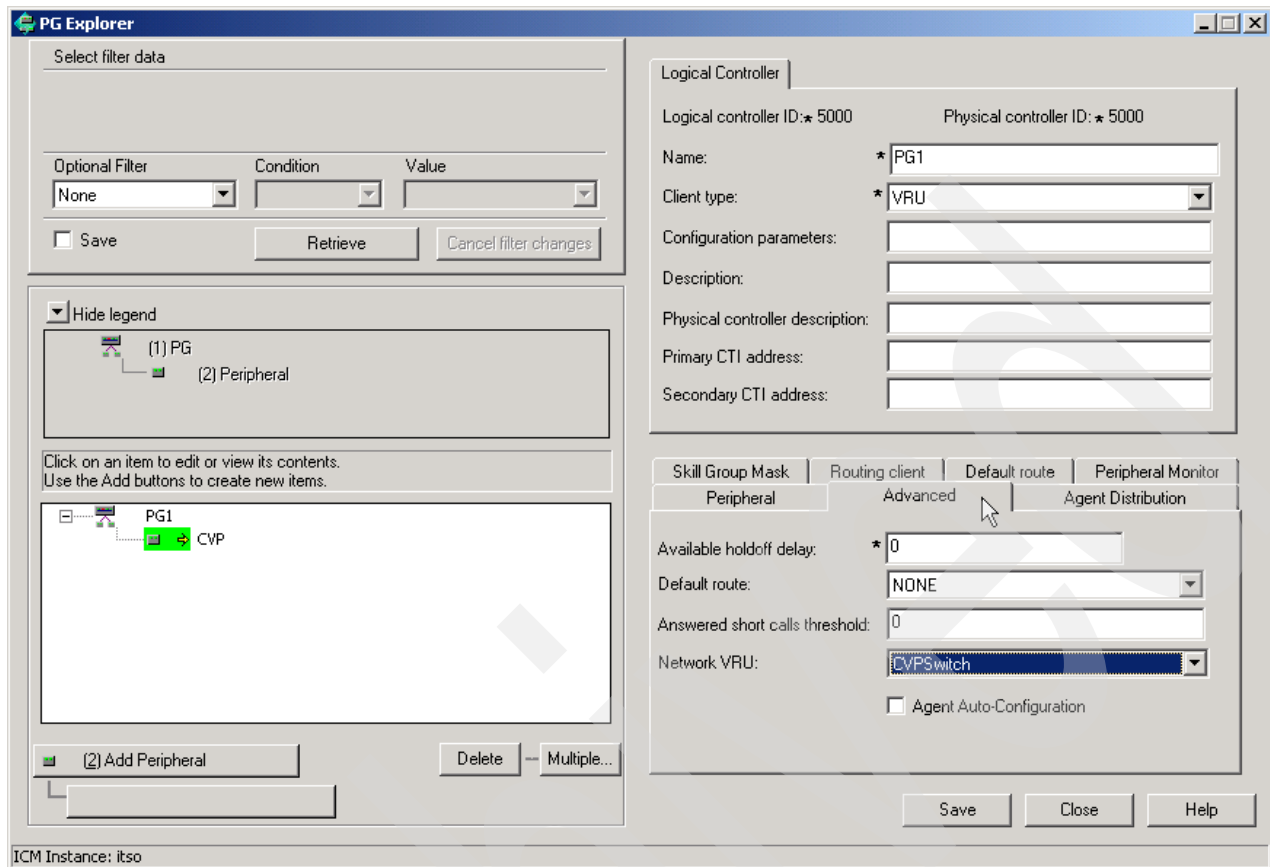


Figure 3-43 Associating the CVPSwitch Network VRU to the Cisco CVP Peripheral

Now, we create a single *Network VRU Script* that all of our ICM routing scripts can use to invoke applications on the Cisco CVP VoiceXML Server. It is actually a misnomer to call it a Network VRU Script. What we are really doing is providing ICM with the mechanism to invoke a particular Network VRU Script, the one that the Cisco CVP Call Control Server understands to mean, “run a VoiceXML Server application”. All we really need to do is configure this script’s name and identifier.

18. We do so using another ICM Configuration Manager tool, the Network VRU Script List. Double-click **Tools** → **List Tools** → **Network VRU Script List** as shown in Figure 3-44.

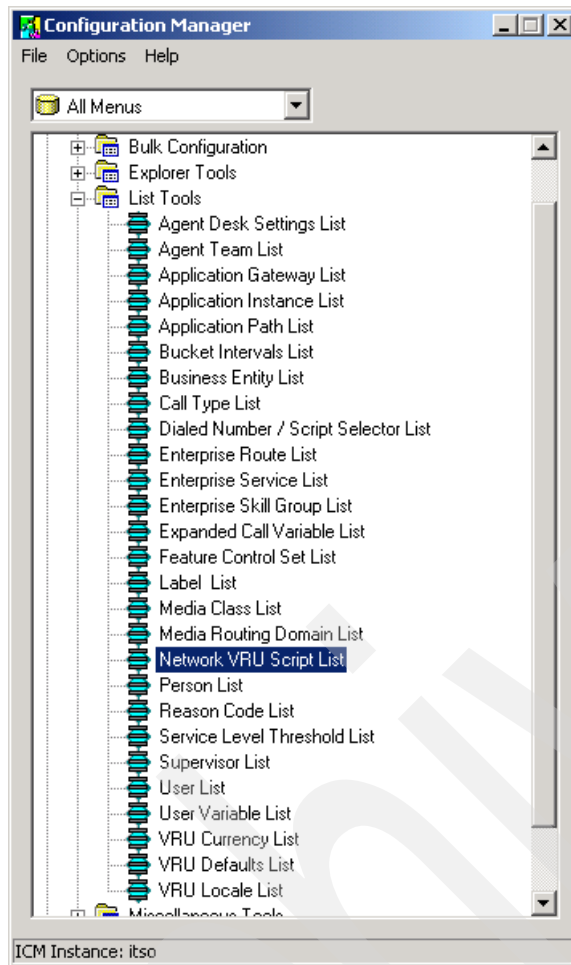


Figure 3-44 Start the Network VRU Script List tool

19. When the Network VRU Script List tool starts, click **Retrieve** to load any existing configuration data. Click **Add**.

20. On the right side of the window, a new pane opens. Set the following values:

- Set Name field value to VoiceXML\_Application (this could be a name of your own choosing).
- Select **CVPVRU** from the Network VRU menu list.
- Set the VRU script name to GS,Server,V (Be extremely careful to copy the character case exactly as shown in Figure 3-47 on page 84).

We will leave the Timeout value at its default of 180 seconds for now, but keep in mind that this needs to be significantly longer than the actual amount of time you expect a VoiceXML application to take. This timeout should only be used to deal with a hung application. It should not be relied upon as part of the normal user interaction process.

21. Click **Save**, and close the window.

There is only one more step we must take in the ICM Configuration Manager. We must create a *Customer Instance*. The Customer Instance was the original way by which ICM service providers could host multiple tenants in a single ICM environment. ICM Hosted Edition is used for that purpose now, but the Customer Instance configuration remains.

Cisco CVP uses the Customer Instance to create a relationship between dialed number and Network VRU. In complex multi-site network deployments, it is sometimes necessary to direct different callers to different Cisco CVP equipment based on dialed number. In our case, we will not be doing that. But, we still need to create one Customer Instance for all Cisco CVP calls.

22. In the ICM Configuration Manager, double-click **Tools** → **Explorer Tools** → **ICM Instance Explorer** tool as shown in Figure 3-45 on page 83.

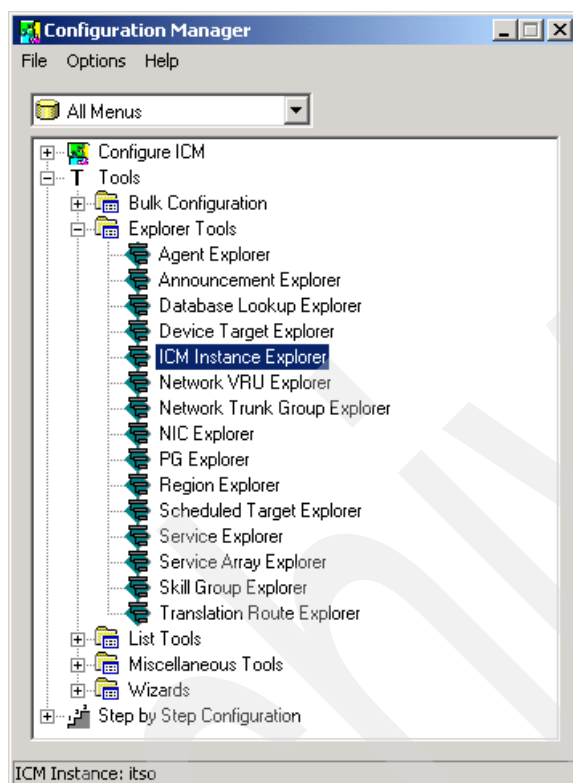


Figure 3-45 Starting the ICM Instance Explorer

23. In the ICM Instance Explorer, click your installation name (in our case it was ITS0) and then click **Add Customer definition**.

24. On the right side, in the Customer Definition tab, enter the following information:

- For Name, enter your company's name. In our case, we used ITS0 again.
- For Network VRU, select **CVPVRU** from the menu list.

25. Click **Save**.

As shown in Figure 3-46 on page 84, we have now associated this Customer Instance with our VoiceXML Gateway's Network VRU. Later, when we configure dialed numbers for individual applications, we will see how to associate those dialed numbers with this Customer Instance, thereby mapping the dialed number to the Network VRU that references our VoiceXML Gateway.

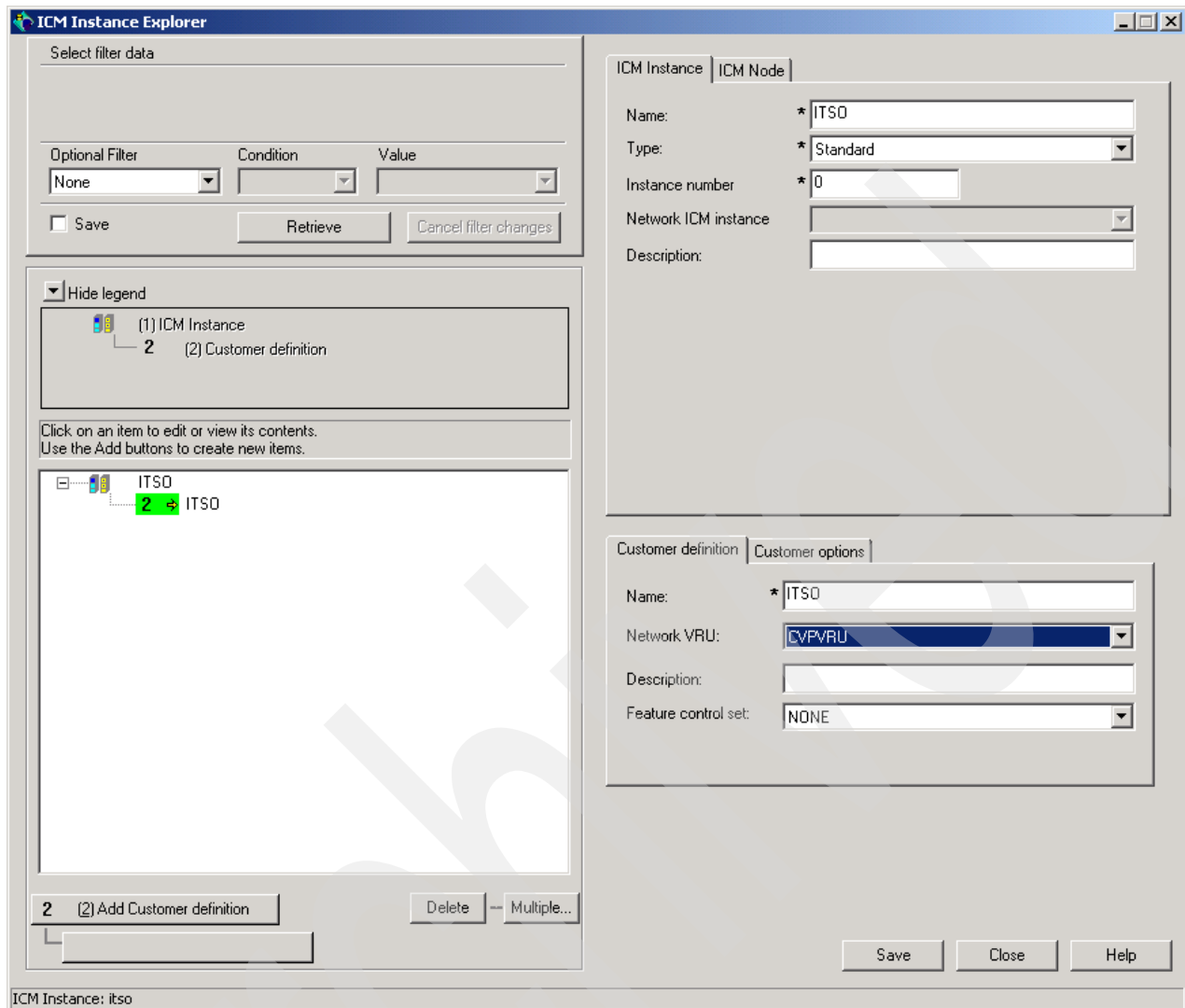


Figure 3-46 ICM Instance Explorer - adding a Customer Instance

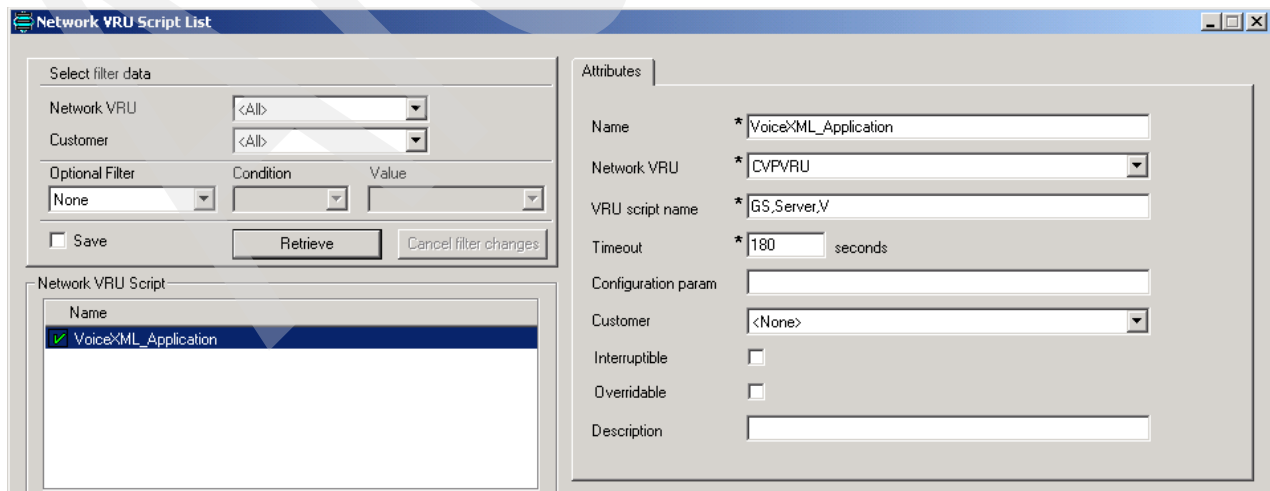


Figure 3-47 Configuring a Network VRU Script to invoke CVP VoiceXML Server applications

26. Now, we are ready to do the necessary work in ICM Setup. In the ICM Admin Workstation folder, click **Setup** (or, navigate to **Start** → **Programs** → **ICM Admin Workstation** → **Setup**).

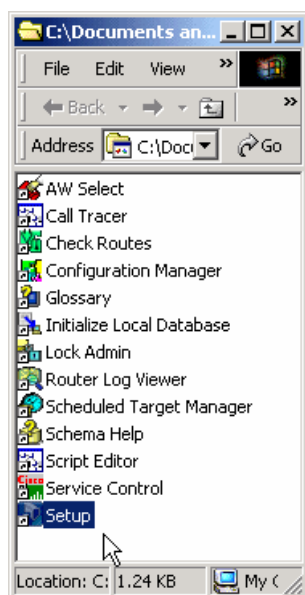


Figure 3-48 Opening the Setup program

27. Notice, as shown in Figure 3-49, PG1A is already installed. This was one of our assumptions from the beginning of this section. If it is not there yet, then you must run Setup from the ICM release disk, and install a Peripheral Gateway component. You can then continue from this point.

Select **PG1A** and click **Edit**.

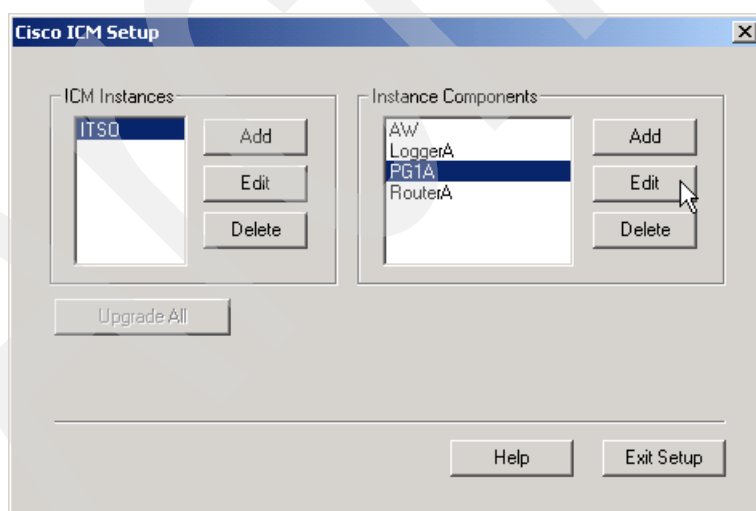


Figure 3-49 ICM Setup main dialog

28. On the Peripheral Gateway Properties dialog box (see Figure 3-50), make sure **PG 1** is selected in the ID menu list and **VRU** appears in the Selected types menu list. Click **Next**.

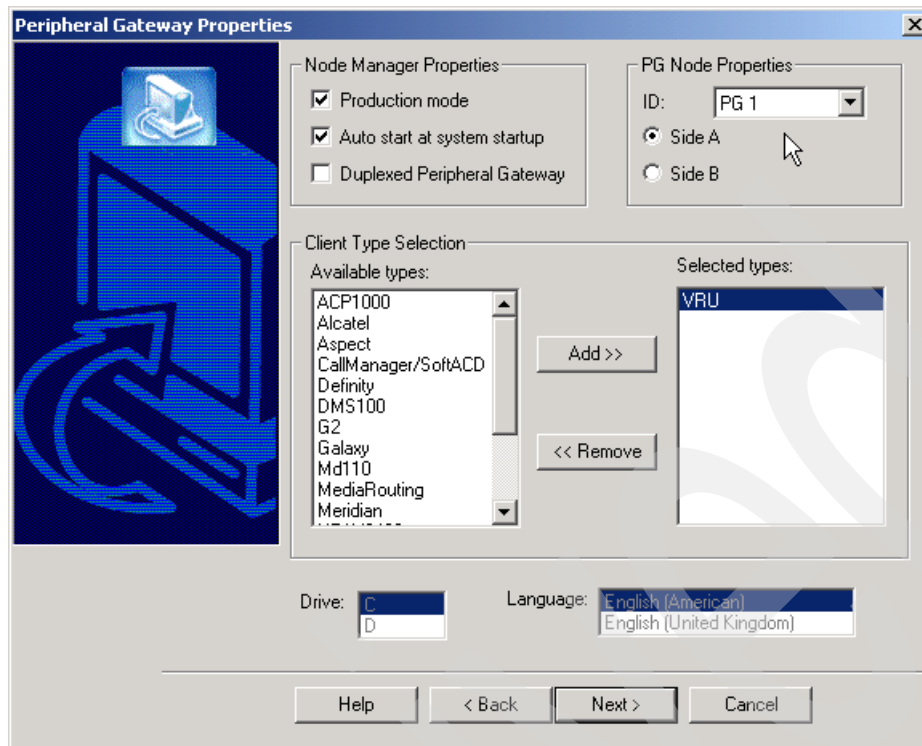


Figure 3-50 ICM Setup - Peripheral Gateway Properties dialog

29.If you receive the message shown in Figure 3-51, click **Yes**.

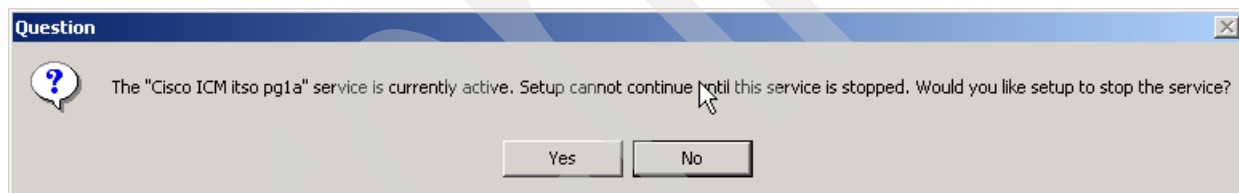


Figure 3-51 ICM Setup - Request to shut down service

30.The Peripheral Gateway Component Properties dialog box opens, as shown in Figure 3-52. Click **Add**.





Figure 3-52 ICM Setup - Peripheral Gateway Component Properties dialog

31. A subdialog box opens as shown in Figure 3-53. Select **VRU** from the Client Type menu list and select the first available PIM, such as **PIM 1** from the Available PIMS menu list. Click **OK**.

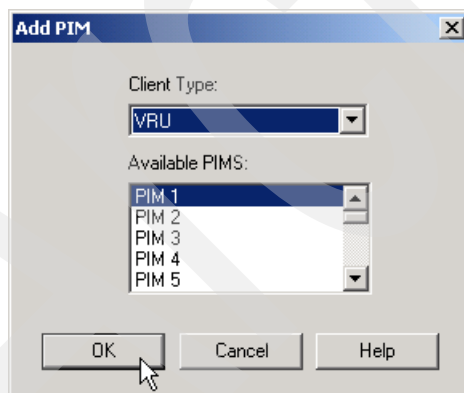


Figure 3-53 ICM Setup - Add PIM subdialog

32. The VRU Configuration dialog box then opens, as shown in Figure 3-54. Here is where we use some of the information we have recorded from earlier dialogs. Make the following changes to the settings:

- Select Enabled
- Set the Peripheral name field value to CVP
- Set the Peripheral ID field value to the Peripheral ID you recorded earlier
- Set the VRU host name field value to the hostname of the CVP Call Control server

- Set the VRU connection port field value to 5000. This matches the VRU Connection Port Number that we left at its default value in the ICM Configuration dialog in the Cisco CVP Application Administration.

Click **OK**.

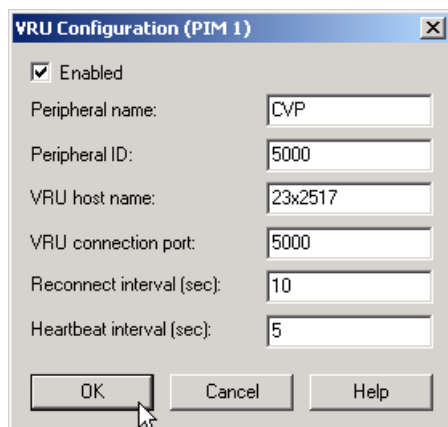


Figure 3-54 ICM Setup - VRU Configuration dialog

Figure 3-55 on page 88 shows how the ICM Setup Peripheral Gateway Component Properties looks now. Notice that we now have an entry under Peripheral Interface Managers.

33. Set the Logical Controller ID field value to the Logical Controller ID that we recorded earlier and ensure that **Service Control** and **Queue Reporting** are selected.

Click **Next**.

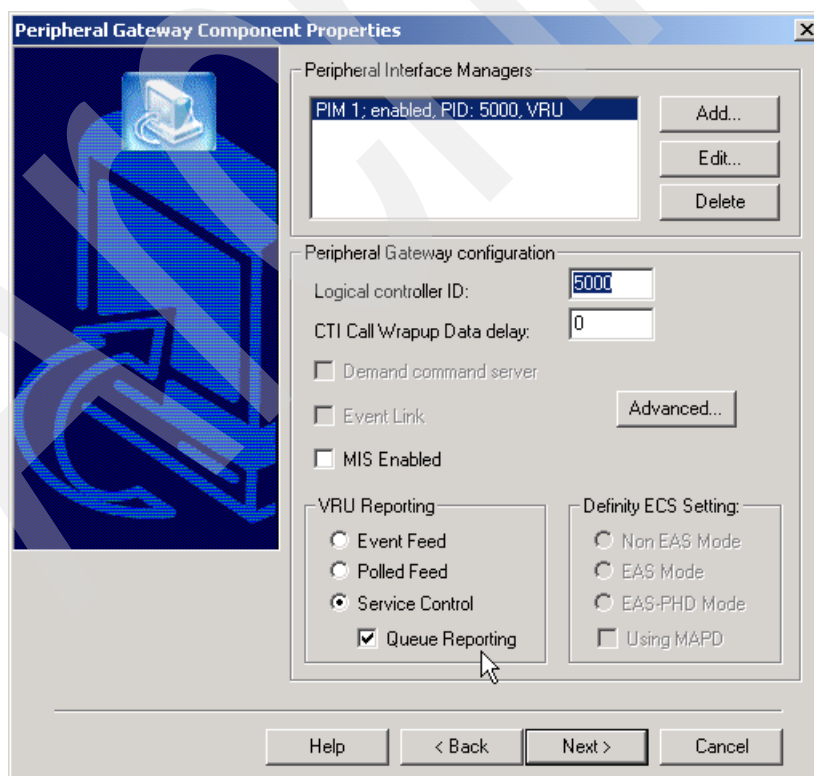


Figure 3-55 ICM Setup - Peripheral Gateway Component Properties, after PIM setup

34. In the next three dialog boxes shown in Figure 3-56 on page 89, Figure 3-57 on page 89, and Figure 3-58 on page 90, accept all the default values and click **Next**.

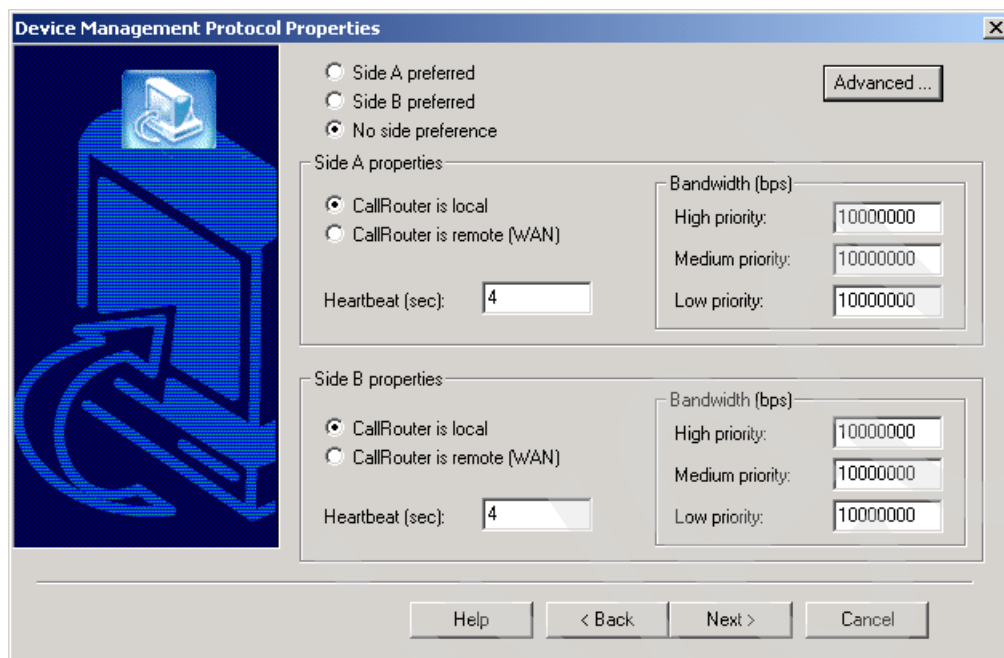


Figure 3-56 ICM Setup - Device Management Protocol Properties

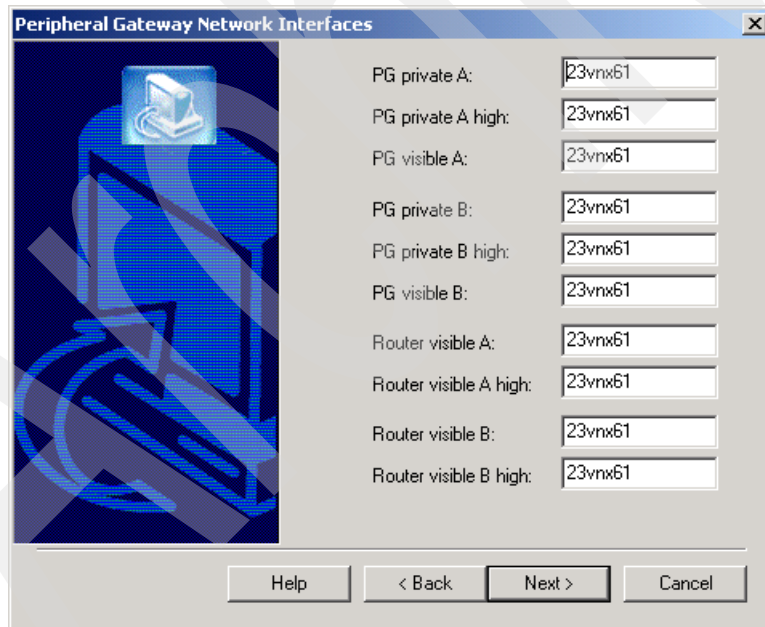


Figure 3-57 ICM Setup - Peripheral Gateway Network Interfaces

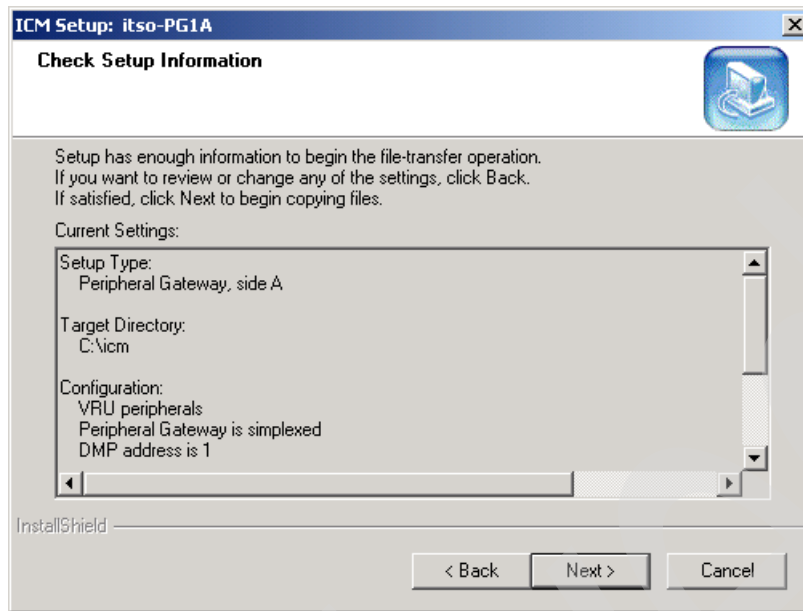


Figure 3-58 ICM Setup - Summary information

35. ICM Setup finally completes its work. In the Setup Complete dialog shown in Figure 3-59 on page 90, allow the program to restart the ICM Node Manager, and click **Finish**. On the main Cisco ICM Setup screen that follows, click **Exit**.

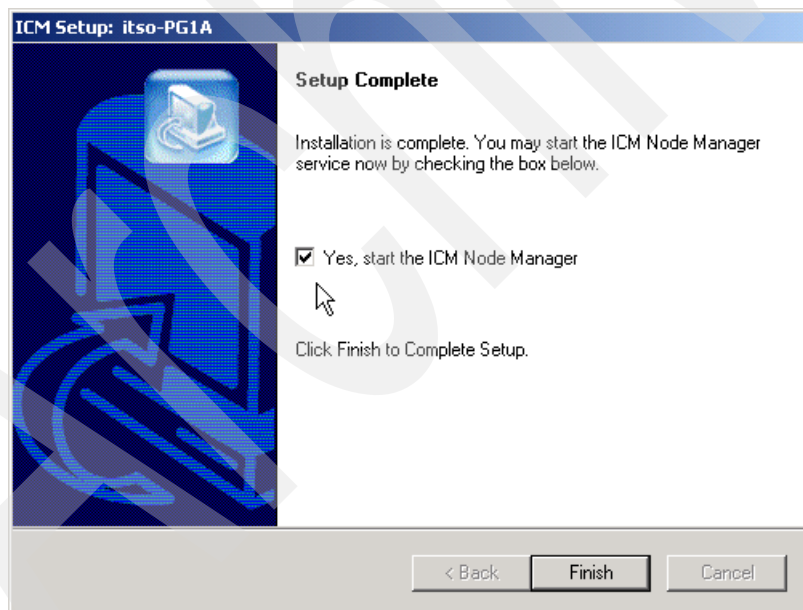


Figure 3-59 ICM Setup complete

36. Within about 60 seconds, ICM automatically connects to the Cisco CVP Call Control server. To verify that, Navigate to **Start** → **Programs** → **Cisco Internet Service Node** → **Application Server** → **Application Administrator** on the CVP Call Control Server machine. Click **Engine**. The dialog box shown in Figure 3-60 on page 91 should open. Notice that the ICM Subsystem now indicates that it is *RUNNING*.

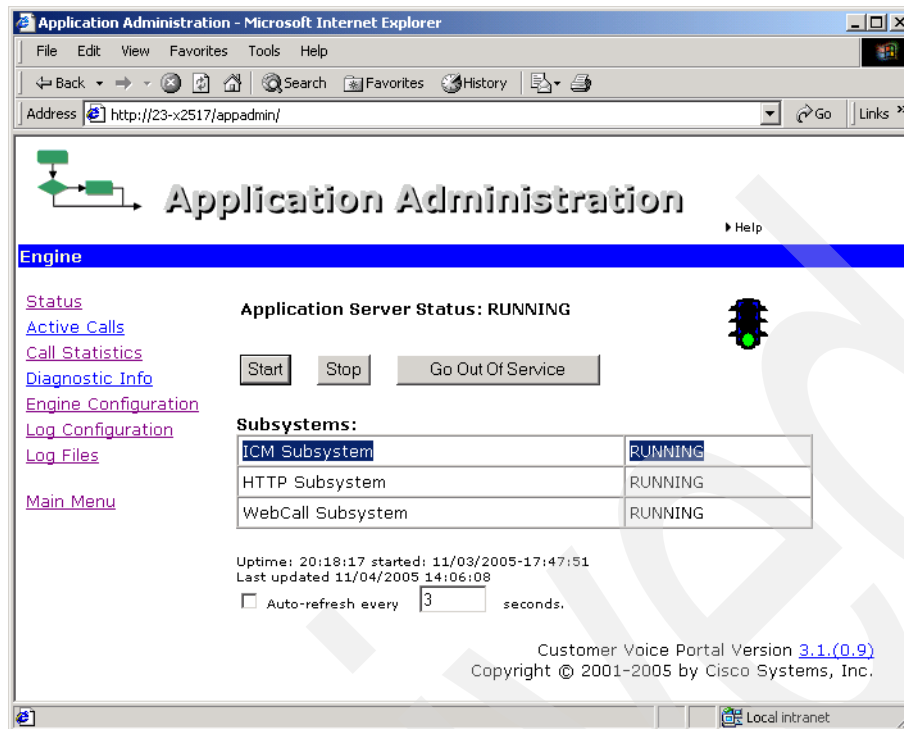


Figure 3-60 Cisco CVP Engine Status screen with ICM connected  
Screen capture reprinted by permission from Microsoft Corporation

### 3.3 Overview of the WebSphere Edge Server Load Balancer component V5.1.1.41

The Load Balancer component of WebSphere Edge Server adds the ability to balance the load between the WebSphere Voice Server systems in the deployment. It also provides failover, recovery, serviceability, scalability, and allows for weighting of work amongst low and high-end servers all seamlessly.

The Load Balancer software is lightweight and can run on a low-end server.

Each Load Balancer can control multiple clusters. A *cluster* is group of application servers that collaborates for the purposes of workload balancing and failover. Within a cluster, each server must have the same abilities. For example:

- ▶ Each cluster can only contain WebSphere Voice Servers running the same languages. Support for multiple languages has been introduced in WebSphere Voice Server V5.1.3.
- ▶ Each server within a cluster must have the same engines installed. That is, they can have both ASR and TTS, just TTS, or just ASR for the same languages. There cannot be a mixture of machines, some having ASR and some having TTS.

The Load Balancer can be configured to have multiple clusters. Each cluster is assigned an IP address, which is then used by the IVR to direct the MRCP requests. When the IVR sends an MRCP message, it is directed to the Load Balancer's dispatcher. The dispatcher then assesses the servers in the cluster and selects an available server with the least load. The message is forwarded on to this server. In a voice server configuration, the Load Balancer uses Media Access Control (MAC) based forwarding, and so the voice server responds to the

message directly back to the IVR. The Real Time Protocol (RTP) traffic containing the voice data is sent on a direct connection between the IVR and the voice server, after the Load Balancer has established the voice server to be used.

**Note:** In our lab, only one cluster was defined.

### 3.3.1 Load Balancer with WebSphere Voice Server

This section contains installation instructions for installing the Load Balancer component on a Microsoft Windows 2003 Server. It then explains how to configure Load Balancer to work with multiple WebSphere Voice Servers.

Load Balancer works with groups of WebSphere Voice Server systems called *clusters*. A *cluster* is a logical grouping for Load Balancer to distribute work among all the WebSphere Voice Server systems in the cluster. Any server in the *cluster* must be able to handle the request. Our *cluster* address is 9.42.171.89. The *cluster* address is a secondary IP address that the Load Balancer machine listens to and is not assigned to any other machine in the network. It is not the primary IP address of the Load Balancer but simply an extra address that the Load Balancer will listen to.

The Cisco router references the *cluster* address rather than a specific WebSphere Voice Server system address. This is done by specifying the two configuration parameters shown in Example 3-7.

*Example 3-7 Configuring the Cisco router to use the WebSphere Voice Server cluster address*

---

```
ivr asr-server rtsp://9.42.171.89/media/recognizer
ivr tts-server rtsp://9.42.171.89/media/synthesizer
```

---

These parameters were set in both Cisco routers used. The AS5400HPX Universal Gateway and the Cisco 1751-V Modular Access Router.

### 3.3.2 Hardware and software prerequisites

For Microsoft Windows Server 2003 the minimum requirements are:

- ▶ Intel Pentium® III processor (minimum 1 GHz or equivalent processor). Load Balancer can run on a low-end server.
- ▶ 50 MB of available disk space for installation, plus extra for logs
- ▶ Java2 Runtime Environment, Java V1.4.2.
- ▶ The following Network Interface Cards (NICs) are supported:
  - 100 Mb Ethernet
  - 1 Gb Ethernet
  - Multi-port Ethernet NICs

**Restriction:** The Windows version of Load Balancer cannot be installed on the same server as IBM Firewall.

### 3.3.3 Microsoft Windows Server 2003 installation

The ITSO lab had a stand-alone version of the Load Balancer. This meant the Java runtime environment had to be installed and configured beforehand. We used the IBM Java Runtime Environment (JRE) V1.4.2. This can be downloaded from the internet.

When it is installed, ensure the following Windows Environment variables have been configured:

- ▶ Include the following in the Path environment variable:

`C:\xx\java\jre\bin`

- ▶ Create a new environment variable called JAVA\_HOME and set its value to:

`C:\xx\java\jre`

Where xx is the directory the JRE was installed in.

- ▶ Reboot the server

To install Load Balancer, follow the steps in *IBM WebSphere Voice Server for Multiplatforms V5.1.1/V5.1.2 Handbook, SG24-6447*. For WebSphere Voice Server V5.1.3, the provided Fix Pack V5.1.1.41 needs to be installed after installing Load Balancer. Run the self-installing executable file:

`eLBW2K-5.1.1.41.exe`

Follow all the installation instructions. This fix pack will uninstall previous versions of Load Balancer before installing the updated version.

### 3.3.4 Configuring the Load Balancer system

The configuration of Load Balancer is a complicated process. Read this section completely before starting the configuration process.

#### Network Prerequisites

Complete the following tasks or obtain the information requested before continuing:

1. The Load Balancer requires a static, nonforwarding address (NFA).
2. Each WebSphere Voice Server to be added to the Load Balancer cluster must be fully installed and verified. Each server must have a static IP address configured.
3. Obtain a static IP address to use as the cluster address.  
Ping the cluster IP address to make sure that it is not in use. Every ping *must* time out!
4. Ping the Load Balancer from the IVR. Every ping *must* succeed.
5. Ping each WebSphere Voice Server from the Cisco router. Every ping *must* succeed.
6. Ping each WebSphere Voice Server from the Load Balancer. Every ping *must* succeed.
7. Ping the Cisco router from each WebSphere Voice Server. Every ping *must* succeed.

Only proceed if all these steps pass and there is no suggestion of duplicate responses or lost packets. Otherwise, troubleshoot your network before configuring Load Balancer.

**Attention:** In our lab, we had to put each of the servers host name and IP address into the Windows HOSTS file. This was due to the lab DNS server had not been configured with all the host names and, therefore, was not resolving host names correctly.

#### Configuring the network

The following steps explain the configuration for a single cluster on the Load Balancer to match our ITSO configuration:

1. Set up your WebSphere Voice Servers so that they are on the same LAN segment.  
Ensure that network traffic between the machines does not have to pass through any routers or bridges.

2. Ensure the network adapters of the workstations are all correctly configured. A loopback device needs to be added to each WebSphere Voicer Server, with the cluster IP address. Table 3-1 describes how to do this.

*Table 3-1 Commands to alias the loopback device for Load Balancer*

<p>On Microsoft Windows Server 2003</p>	<ol style="list-style-type: none"> <li>a. If you have already added a MS Loopback Adapter Drive go to step i.</li> <li>b. Click <b>Start</b>, then click <b>Control Panel</b>.</li> <li>c. Click <b>Add Hardware</b>. This launches the Add Hardware Wizard. Click <b>Next</b>.</li> <li>d. Select <b>Yes, I have already connected the hardware</b>, then click <b>Next</b>.</li> <li>e. If the MS Loopback Adapter is in the list, it is already installed. Click <b>Cancel</b> to exit.</li> <li>f. If the MS Loopback Adapter is not in the list, select <b>Add a new hardware device</b> and click <b>Next</b>.</li> <li>g. Select <b>Install the hardware manually from a list</b>, and click <b>Next</b>.</li> <li>h. Select <b>Network adapters</b> and click <b>Next</b>.</li> <li>i. On the Select Network Adapter panel, select <b>Microsoft</b> from the Manufacturers menu list, then select <b>Microsoft Loopback Adapter</b>. Click <b>Next</b>.</li> <li>j. Click <b>Next</b> to start the installation.</li> <li>k. Click <b>Finish</b> to complete installation.</li> <li>l. From the Control Panel, double-click <b>Network Connections</b>.</li> <li>m. Select the connection with Device Name <b>Microsoft Loopback Adapter</b>.</li> <li>n. Right-click and select Properties from the context menu list.</li> <li>o. Select <b>Internet Protocol (TCP/IP)</b>, then click <b>Properties</b>.</li> <li>p. Click <b>Use the following IP address</b>. Enter the IP address using the cluster address and Subnet mask using the subnet mask of the WebSphere Voice Server. Leave the default gateway blank. Enter 127.0.0.1 as the Preferred DNS Server.</li> <li>q. Click <b>OK</b>.</li> <li>r. Click <b>Close</b> in the "Connection Properties" dialog.</li> </ol>
---	--



**Important:** The binding order of the network adapters is important. The real network adapter must be higher than the loopback adapter. To change this, perform the following actions:

- ▶ Open Network settings folder.
- ▶ Navigate to **Advanced** → **Advanced Settings**
- ▶ Ensure the real (physical) network adapter is at the top and the loopback adapter is below it.

Figure 3-61 shows the Advanced settings with the adapters configured correctly.

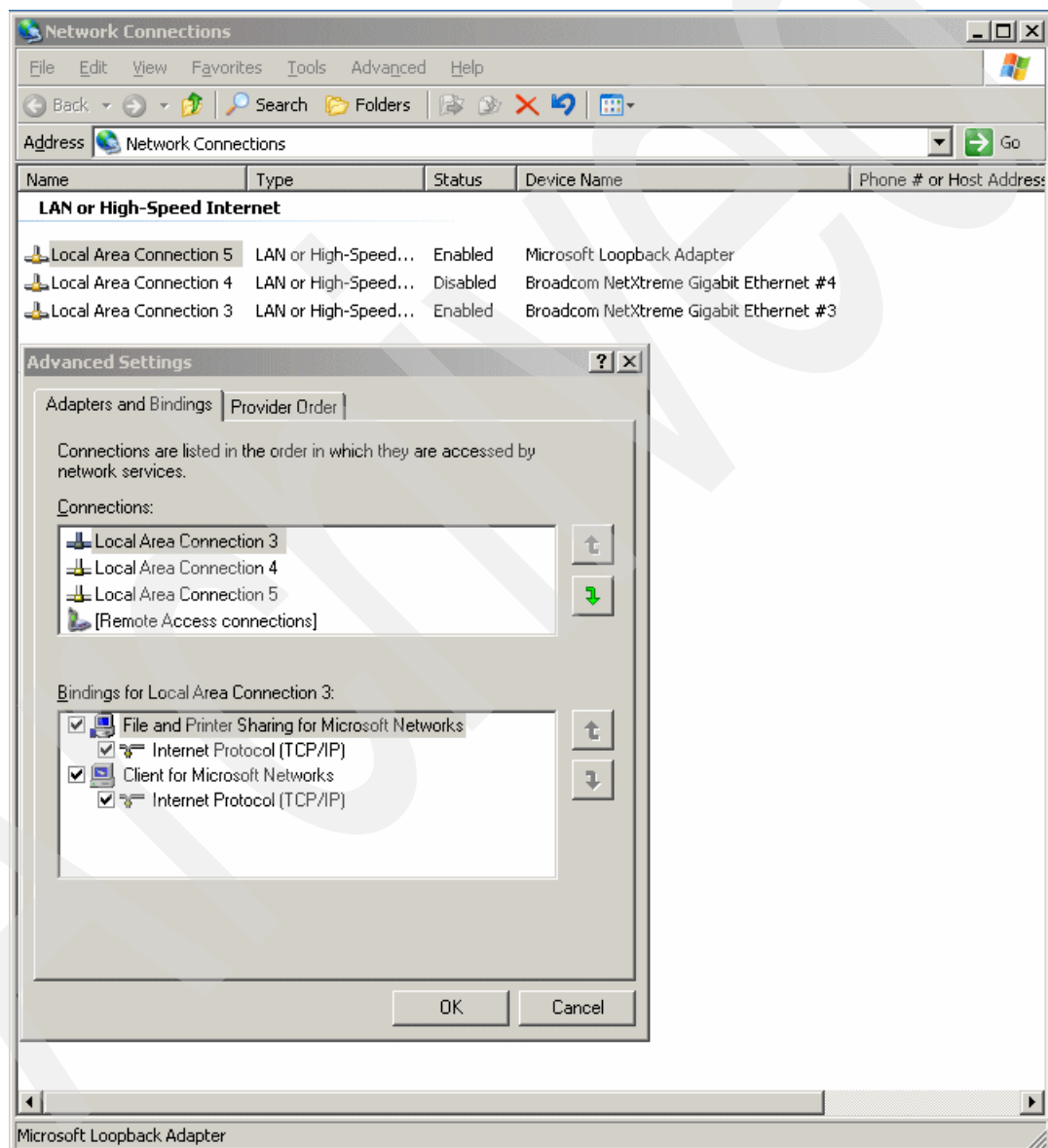


Figure 3-61 Advanced adapter bindings  
Screen shot reprinted by permission from Microsoft Corporation

## Configuring Load Balancer

To configure the Load Balancer a series of configuration objects are created. These configuration objects are detailed below in the order that they must be created.

In the ITSO setup, the configuration used one Load Balancer and three WebSphere Voice Servers are each identical to the other. That is the two languages, US English and Canadian French, with both Automatic Speech Recognition and Text to Speech functions for each language. Table 3-2 shows the ITSO configuration.

Table 3-2 ITSO Network configuration

Server	Name	Language	IP address
Load Balancer	23vnx79.itso.ral.ibm.com		9.42.171.120
Voice Server	bc1srv5.itso.ral.ibm.com	US Eng / Ca Fr	9.42.171.169
Voice Server	bc1srv8.itso.ral.ibm.com	US Eng / Ca Fr	9.42.171.98
Voice Server	bc1srv9.itso.ral.ibm.com	US Eng / Ca Fr	9.42.171.97
Cluster address			9.42.171.89

To configure the Load Balancer:

1. Connect to the Load Balancer Host. In our configuration, this was our Load Balancer hostname, 23vnx79.
2. For the Host, start the Executor. In our configuration, this was our Load Balancer IP address, 9.42.171.120
3. For the Executor, add the cluster. In our configuration this was our cluster address, 9.42.171.89. The cluster address or IP address is used by the Cisco router to direct the MRCP requests.

**Note:** During this installation the name of the cluster was set to the IP address. If another name was used for the cluster the configuration seemed to fail.

4. For the Cluster, add a well known port. According to the MRCP specification, the default well known port is port 554. The Cisco router and WebSphere Voice Server use port 554 as the default.
5. For the Port, add the servers to receive work. In our configuration, we had three WebSphere Voice Servers and they were named by their hostnames, bc1srv5, bc1srv8, and bc1srv9.
6. For the Host, start the Manager and then start Advisor.
7. For the Advisor, configure a Ping Advisor on port 554. Advisors are used by the Load Balancer to insure that server machines are online and able to handle requests. An advisor will send a special request (similar to a ping) to the specified port on each WebSphere Voice Server machine in the list.

These configuration steps are explained in detail in *BM WebSphere Voice Server for Multiplatforms V5.1.1/V5.1.2 Handbook, SG24-6447*. Figure 3-62 on page 97 shows the completed Load Balancer configuration as displayed by the Load Balancer Console. This is for the ITSO complex environment. You can see the three WebSphere Voice Servers in cluster 9.42.171.89.

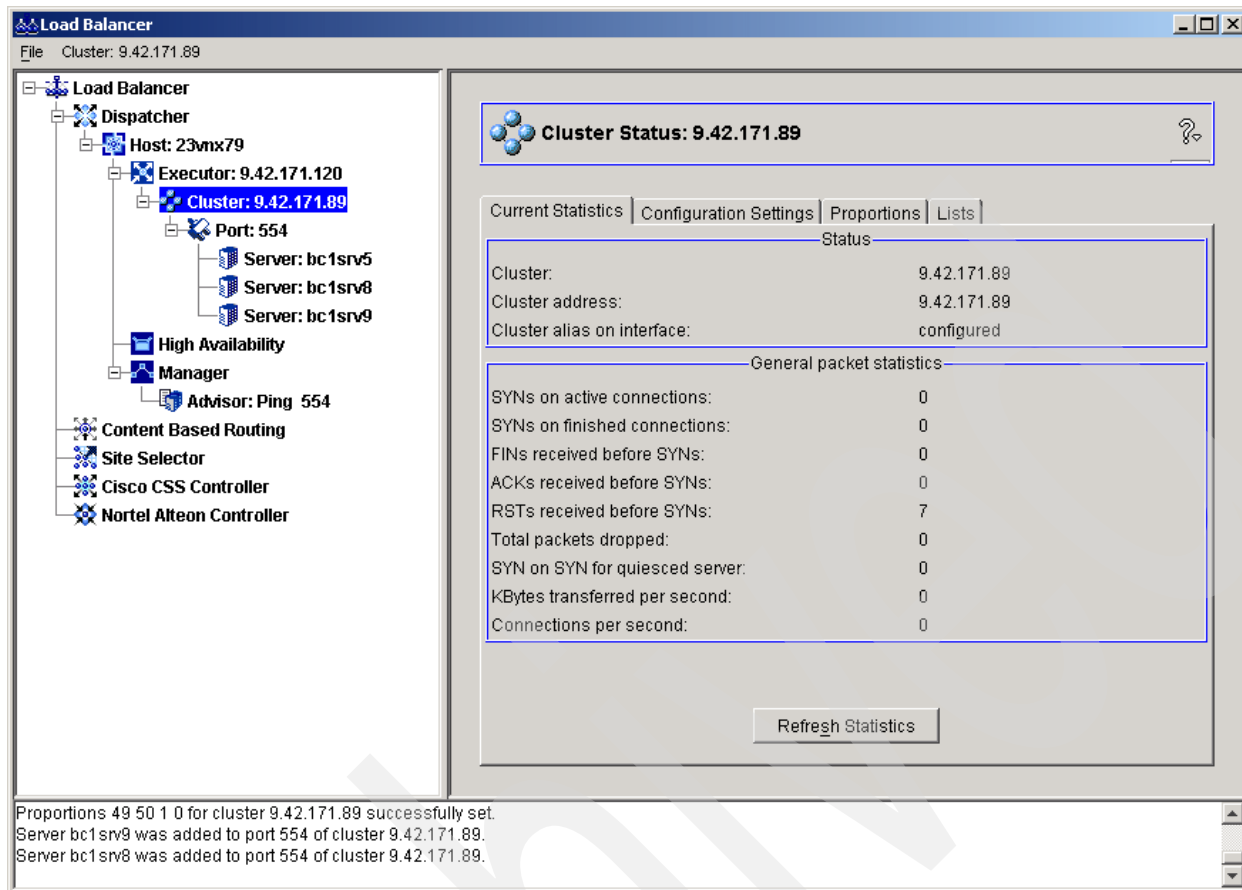


Figure 3-62 Load Balance console

Example 3-8 shows a portion of the actual Windows HOSTS file used on the Voice Servers. Note how all the host names in the cluster are resolved as well as the Load Balancer.

*Example 3-8 Sample portion of the HOSTS file*

9.42.171.120	23vnx79
9.42.171.169	bc1srv5
9.42.171.98	bc1srv8
9.42.171.97	bc1srv9
127.0.0.1	localhost

To test the load balancer, we made several concurrent calls using the HelloWorld1 application created in Chapter 2, “Basic deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3” on page 13. Figure 3-63 on page 98 show the active monitor connections. This monitor shows how the MRCP requests generated by the router are being load balanced over the three WebSphere Voice Servers in the complex.

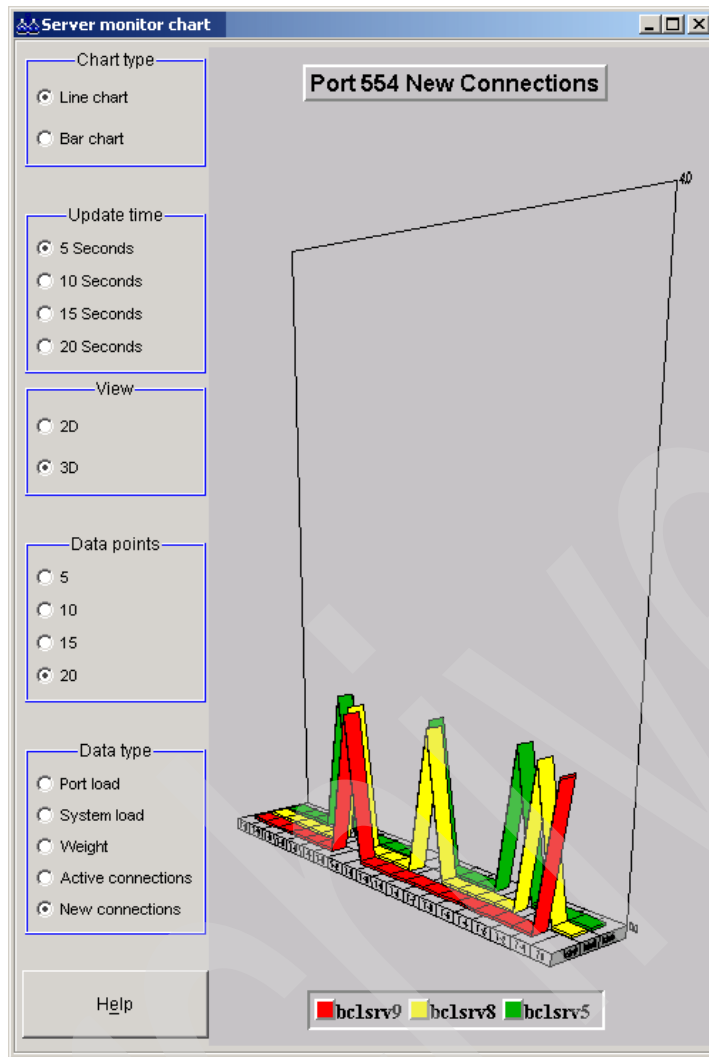


Figure 3-63 Load balancing of MRCP requests

### 3.3.5 Network infrastructure for complex deployment

After all the Cisco CVP components and the WebSphere Voice Server for Multiplatforms V5.1.3 are installed, the system is ready to accept higher volume calls, similar to a small-scale production environment. The Redpaper physical topology can be seen in Figure 3-64.

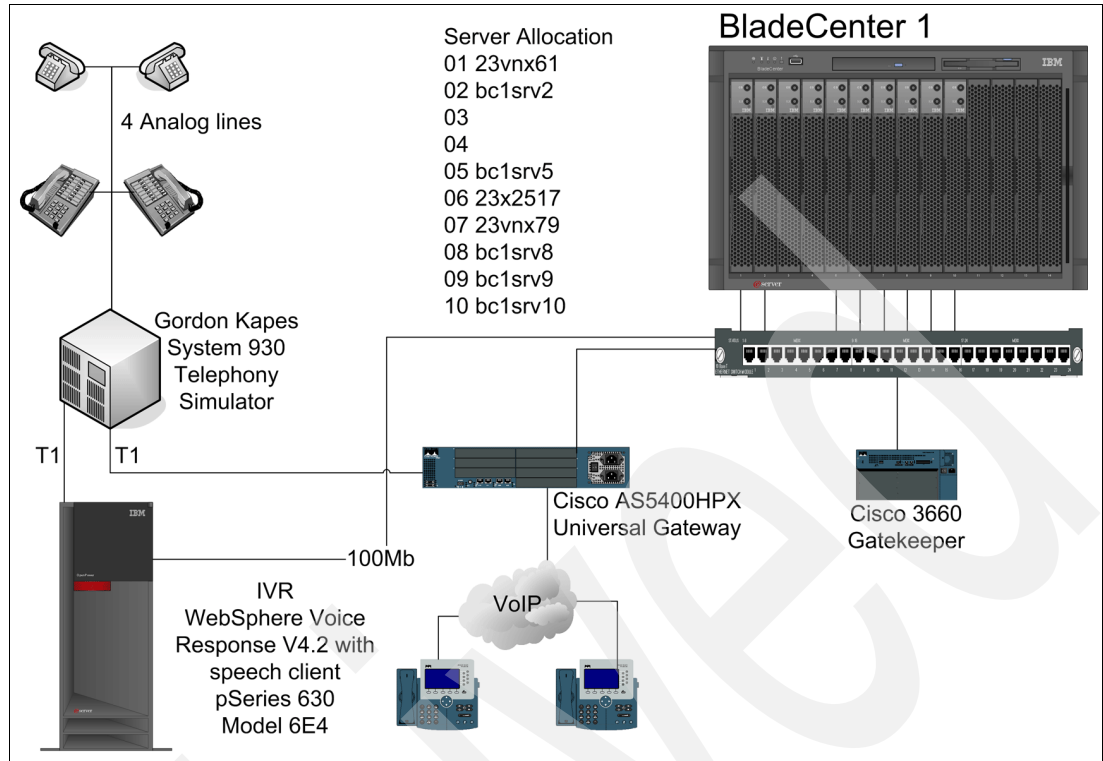


Figure 3-64 Network infrastructure for basic deployment

The IP addresses for all of the components are displayed in Table 3-3.

Table 3-3 IP addresses in complex deployment

Function	Host name	IP address
Cisco CVP V3.1 VoiceXML Server with Tomcat	bc1srv10	9.42.171.24
Cisco CVP V3.1 VoiceXML Server with WebSphere Application Server V5.1.1	bc1srv2	9.42.171.94
WebSphere Edge Server Load Balancer component V5.1.1.41	23vnx79	9.42.171.120
WebSphere Voice Server for multiplatforms V5.1.3	bc1srv5	9.42.171.169
WebSphere Voice Server for multiplatforms V5.1.3	bc1srv8	9.42.171.98
WebSphere Voice Server for multiplatforms V5.1.3	bc1srv9	9.42.171.97
My SQL server 5.0.1.5 Cisco CVP V3.1 Call Control Server	23x2517	9.42.171.150
Cisco ICM 6.0 SR4	23vnx61	9.42.171.106
Cisco AS5400 router	AS5400	9.42.171.128
Cisco 3660 Gatekeeper	Gatekeeper-3660	9.42.171.186
WebSphere Voice Response with speech client V4.2		9.42.171.16

Archived

## Developing voice applications with Cisco CVP V3.1 Studio

This chapter describes how to develop voice applications using Cisco CVP V3.1 Studio. It gives you a general introduction to Cisco CVP VoiceXML technology and an overview of the Cisco CVP architecture. We discuss the various elements in Cisco CVP Studio and give examples on how to use them. We also show how to integrate an application with ICM Call Routing. Lastly, we talk about IBM WebSphere Voice Toolkit V6.0.1 and demonstrate usage of the following:

- ▶ Lexicon pronunciation editor
- ▶ Creating an external grammar file
- ▶ Reusable Dialog Components

## 4.1 General introduction to Cisco CVP VoiceXML technology

*Voice eXensible Markup Language* (VoiceXML) is an IVR programming language that was created for the purpose of bringing the advantages normally associated with Web-based development and content delivery into the often bewildering and byzantine world of Interactive Voice Response application development. VoiceXML, as a comprehensive, standardized markup language found its origins within an AT&T research project called the Phone Markup Language (PML) the goal of which was to simplify and standardize the IVR application development process. Ironically, as AT&T underwent significant corporate reorganization, separate teams at Lucent, Motorola, and AT&T branched off and continued work on separate and often disparate versions of the original PML project, negating the original goal of consolidation and standardization. By 1998 the W3 Consortium began to take a more active hand in the development and standardization of an IVR oriented programming language and by the year, 2000, the VoiceXML V1.0 specification had been submitted to the W3C as the basis of a new international standard for IVR application development. As of this writing, VoiceXML V2.0 is the specification accepted by the W3C and the defacto industry standard for IVR programming.

### 4.1.1 How VoiceXML addresses limitations of traditional IVR technology

VoiceXML was introduced as a means of addressing the glaring limitations apparent in many traditional, proprietary IVR solutions. As a result of ever-heightening customer expectations for rapidity, interactivity and convenience in customer service oriented interactions, businesses require their IVR solutions to be every bit as flexible and dynamic as other types of ubiquitous customer-facing applications deployed throughout the enterprise. As time went on, it became increasingly evident that proprietary IVR solutions were simply not exhibiting the adaptability and scalability required for the rapid integration of new customer-driven business requirements. VoiceXML was introduced as a means of standardizing and simplifying the IVR application development process and has allowed business organizations to reap some very important benefits:

- ▶ Because VoiceXML is a standardized markup language, the need for an extensive, in-depth understanding of proprietary IVR implementations has been eliminated altogether. VoiceXML is designed to leverage existing Web infrastructure and can be delivered over the omnipresent HTTP protocol. Complex, highly scalable IVR applications can now be developed in a markup language that mimics the familiar style and structure of the beloved and ubiquitous hypertext markup that lies behind every HTML page on the World Wide Web.
- ▶ Because VoiceXML has been designed explicitly to leverage existing Web infrastructure, IVR applications written in VoiceXML can integrate and intercommunicate with virtually any existing business application or data infrastructure that supports standard communication and data access protocols. This obviates the need for expensive and time consuming upgrades to back-end systems for the sake of interoperability with the IVR.
- ▶ IVR applications have now been rendered (for the most part) completely portable. As long as the application conforms to current VoiceXML standards, you can be reasonably certain that an application will run successfully on any compliant platform. What this all amounts to is decreased cost of ownership and significant gains in return on investment (ROI) for businesses that commit to implementing their IVR solutions on a platform that complies with up-to-date VoiceXML standards.

#### Where to find official VoiceXML specifications

Further technical information about the VoiceXML standards and specifications can be found:

<http://www.w3.org/TR/voicexml20>



### 4.1.2 Challenges inherent In VoiceXML development

Despite the fundamentally robust nature and broad acceptance of VoiceXML as a standard for IVR programming, there remain a number of challenges inherent in the VoiceXML development process. Some of these challenges are artifacts of the language's roots in web-based markup languages and some arise as a result of the fundamental differences in the delivery of content via a voice-based paradigm versus delivery via more visually oriented paradigms. For example, VoiceXML (much like HTML delivered over HTTP) is a stateless protocol and preservation of data across a user session is an issue that always looms large in the development process. There is also the issue of back-end integration. An enterprise application rarely or never runs in as a standalone entity in a *vacuum* and almost always requires a consistent and reliable means of communicating with back-end systems, such as databases containing customer or product information. VoiceXML itself does not provide a robust data access mechanism out-of-the-box. VoiceXML coding for applications that require extensive and varied user interactions can be dauntingly verbose and complex. Furthermore, many applications require the dynamic insertion of content based on data that is available exclusively at run time which creates an entirely new need, some type of VoiceXML application server in the spirit of a dynamic HTTP server, such as Apache, Tomcat, IIS, and so forth, that is capable of serving dynamically generated VoiceXML code and further, a framework for developing the applications that will generate this dynamic VoiceXML code.

### 4.1.3 How Cisco CVP addresses VoiceXML development challenges

In order to address some of the issues addressed above, Cisco has developed a full featured VoiceXML development and deployment suite that allows for the creation of robust, complex applications with the ability to integrate seamlessly with back-end enterprise systems. The crown jewel in the CVP application development suite is the GUI based integrated development environment (IDE). It provides a dizzying array of reusable and fully configurable components that allow developers to rapidly create, deploy, and administer fully functional VoiceXML based IVR applications in an elegant drag and drop environment. There is not even a need for in-depth knowledge of the VoiceXML programming language. The use of Cisco's VoiceXML studio environment allows IVR developers to capitalize on the following essential benefits:

- ▶ Rapid application development
- ▶ Design and build simultaneously
- ▶ Develop applications using an intuitive graphical interface
- ▶ Little need for extensive knowledge of VoiceXML, Java, or XML

The VoiceXML Studio development environment is used to develop the Java applications that will dynamically generate standards compliant VoiceXML code at runtime and works in tandem with the Cisco CVP VoiceXML application server. The Cisco CVP VoiceXML Server is a powerful J2EE and J2SE compliant runtime engine that allows for full caller-driven interaction with deployed IVR applications. Cisco CVP VoiceXML server performs the following essential functions:

- ▶ Dynamic generation of VoiceXML code
- ▶ Integration with back-end systems
- ▶ Session management
- ▶ System administration and user management

## 4.2 Overview of Cisco CVP Solution Architecture

The following is a more in-depth description of some of the key features and benefits associated with the components of the Cisco CVP framework.

## 4.2.1 Cisco CVP VoiceXML Studio

*CVP VoiceXML Studio* is an Eclipse-based platform for the development, management, and deployment of sophisticated, feature-rich VoiceXML applications. No prior knowledge of the Eclipse framework is required to use CVP Studio. It is merely the framework that encapsulates the VoiceXML development plug-in. CVP studio is a graphical development environment that provides an intuitive means of developing all types of IVR applications using a top-down design paradigm. Overall application logic is defined in the form of a graphical callflow, which is essentially a flow chart outlining what actions will occur in the application. The beauty of this approach is that application logic can be defined at a very high level without having to immediately delve into the nitty-gritty of how application logic will be implemented. Because of this approach, it is possible for complex, feature-rich applications to be mapped out by designers without any in-depth technical knowledge. This allows for the distribution of application development tasks within teams with diverse skill sets and specializations. The following are some of the specific benefits associated with the CVP Studio plug-in:

- ▶ **Intuitive Interface**

Using a process very familiar to users of flowcharting software such as Microsoft Visio®, users can define the business logic for an IVR application rapidly in the form of a graphical call flow diagram. The call flow is constructed in a simple and intuitive drag-and-drop environment from a powerful toolkit of reusable and fully configurable elements that provide the functionality required by the vast majority of IVR applications. Any functionality that does not exist in these prefabricated elements can be introduced easily by extending an existing element with a well engineered Java or XML API. Entirely new custom elements can be created to meet the business requirements of the application.

- ▶ **Design and build simultaneously**

The studio environment, in addition to being a design tool, also provides the means to rapidly build and deploy a finished application to the Cisco CVP server that allows for a continual cycle of building, deployment, and testing.

- ▶ **Minimal technical knowledge requirements**

Because the studio environment places a robust layer of abstraction between the application developer and the actual low-level implementation details of VoiceXML coding, robust, full-featured IVR applications can be developed and deployed by personnel with little or no technical knowledge of VoiceXML, Java, or XML coding.

- ▶ **Rapid application development**

The time taken to develop a full-featured IVR application in the graphical studio environment can be up to 90% less than what it would take to develop the application as *flat* or *static* VoiceXML documents.

## 4.2.2 Cisco CVP VoiceXML Server

Cisco CVP VoiceXML Server provides back-end integration, session management, dynamic application generation, system management, and user management, which we now briefly describe.

### **Back-end integration**

Cisco CVP server is based on a J2SE/EE framework and consequently provides the developer with access to a wide range of commonly available middleware that allows for full and seamless integration between a customer-facing VoiceXML application and back-end enterprise data repositories.

## Session Management

Call and user data are tracked and preserved automatically with ANI, DNIS, session id, and so on, by the server and can be accessed easily for use in the application.

## Dynamic application generation

Content and application logic is defined in the application developed in the studio environment and then relevant details are filled in the form of dynamically generated VoiceXML at runtime by the server. This allows for almost infinite flexibility in terms of customizing the caller experience.

## System management

Cisco CVP Server provides a full suite of application and system management utilities in the form of canned administration scripts making application deployment and management simple and time effective. Furthermore, common administrative tasks can be performed while the server is in service without disruption to active sessions.

## User management

Cisco CVP Server includes a lightweight customer data management system for use with applications where more robust means for data management are not available. This allows for personalization of caller experience without the need for deployment and configuration of an extensive back-end data management infrastructure.

## 4.3 Elements in studio

The Cisco CVP Studio environment possesses a large collection of fully tested, standards-compliant, prefabricated building blocks with the express purpose of providing commonly required application functionality in *canned*, or prebuilt, form in order to speed application development. These reusable elements encapsulate most, if not all, commonly required functionality and eliminate the need to constantly reinvent the wheel and allow developers to focus on the core task of creating robust and user-friendly voice applications. Reusable elements can be configured easily by the developer to tailor their output to the needs of the application in question. Predefined configurations based on well-known and tested design techniques are provided with each configurable element as a further means of accelerating the application development process. Furthermore, elements are designed so that their placement in the call flow will result in the generation of browser compliant VoiceXML code when the application is run on the Cisco CVP server. Before going any further let us briefly examine some of the key features of these reusable elements.

### 4.3.1 Exit states

Each element employed in a given call flow can be considered to be the equivalent of a *black box* system. The component accepts input, performs some action and then exits passing control to the next element in the call flow. The causal linkage between elements is defined by the *exit state* associated with the element in question. For example, we might have a *string\_parser* element that accepts a string as an input. This input is taken by the *string\_parser* and split into distinct lexical tokens wherever the “-” character is encountered in the input string. The *string\_parser* then stores the tokens in session data. If storage of the tokens is successful, the *string\_parser* element exits with the result of *storage\_successful* and passes control to the next element in the call flow. Alternately, if storage of the tokens fails, the *string\_parser* exits with a *storage\_failed* result and may pass control to a component on a different branch of the call flow (see Figure 4-1 on page 106).

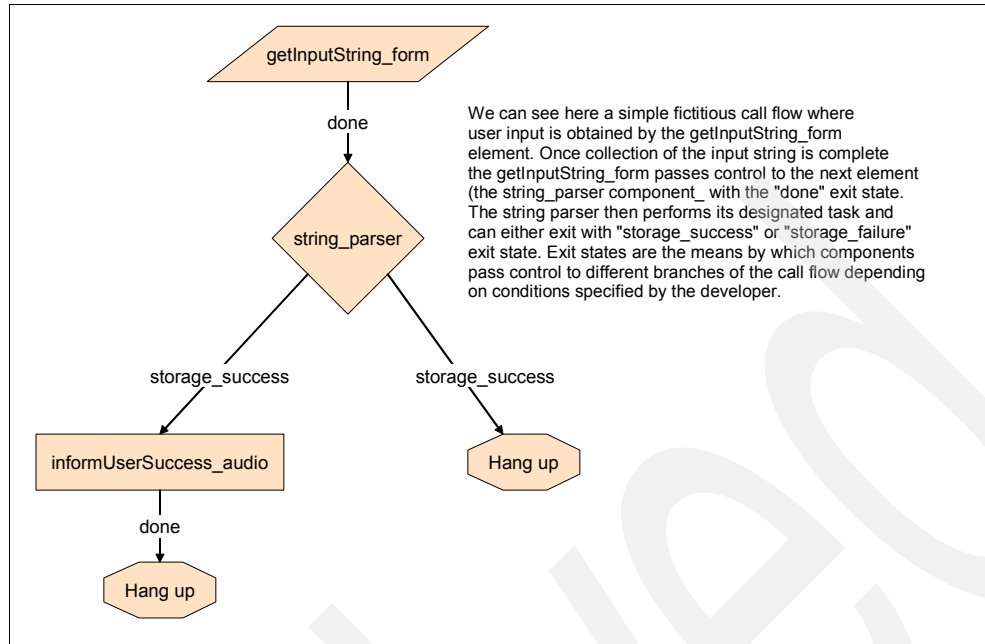


Figure 4-1 Sample call flow diagram

### 4.3.2 Element and session data

Elements in an IVR application are capable of sharing information (communicating) with one another through the means of Session and Element data. Element data is data or variables that exist within the context of the element itself. Element data can be accessed by any other element in the call flow, but can only be modified by the element that created it. Session data is somewhat more flexible than element data. Session data is accessible (readable) by any element within a given call flow and additionally can be modified by any element. Session data exists for the duration of a browser session.

### 4.3.3 Customization

Most elements that can be used in a call flow require a certain amount of customization by the developer so that complex tasks can be performed in a manner appropriate to the application context. There are three types of configurations associated with elements in Cisco CVP studio.

#### Fixed Configuration

A fixed element configuration (see Figure 4-2 on page 107) is maintained in an XML file associated with that component. A fixed configuration is used to maintain a set of constant behaviors and actions that will be performed by the element every time that element is visited in the call flow. This is very handy because not every element in a call flow must perform in exciting, dynamic ways. For example, you might have a simple audio element at the start of the call flow that will always read the prompt Hello and welcome to the stock quotes application when the user's call is connected. There is no need for a fancy API implementation for this type of standardized configuration. Studio will take care of all the details of storing this type of configuration. All the developer needs do is tweak the appropriate setting in the element configuration pane in studio, the rest is transparent.

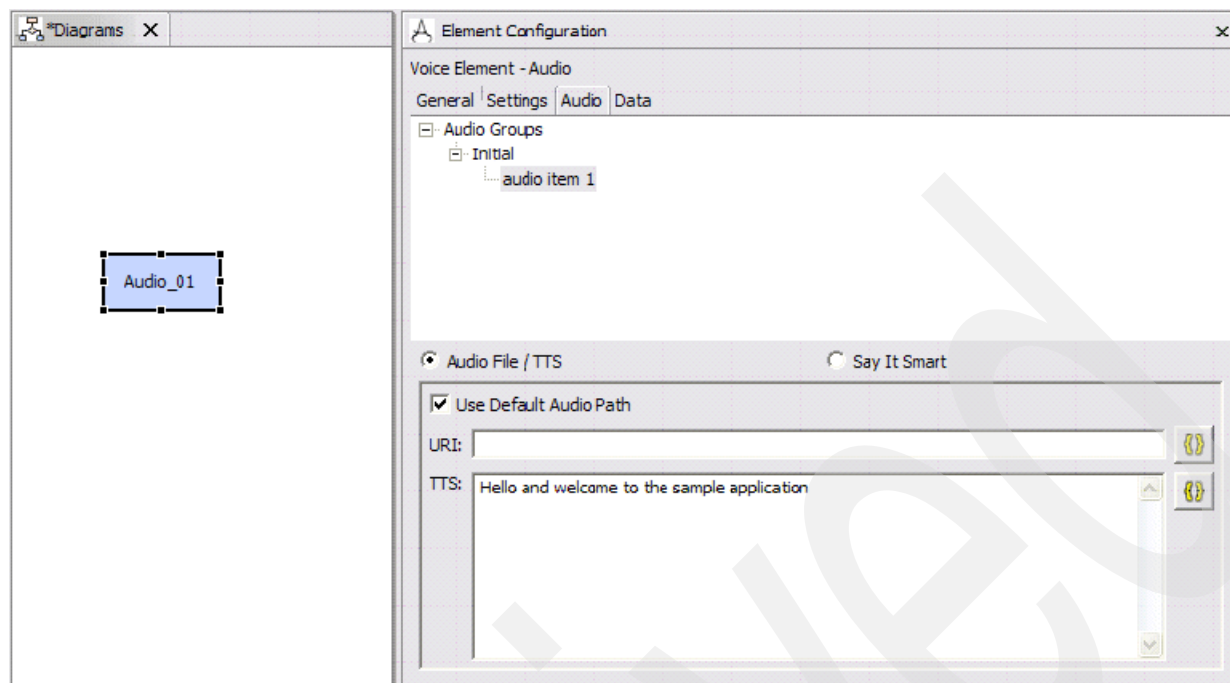


Figure 4-2 An audio element with a fixed configuration, this audio element will always read the TTS string “Hello and welcome to the sample application”

## Java API Configuration

A highly customizable Java API has been defined for the purpose of defining highly specialized customizations to reusable and custom components in the studio environment. This approach to customization is extremely versatile and developers are limited in the range of customizations available basically only by their imagination and level of programming ability. However, this approach does suffer the limitation of having the requirement that the developer have at least a passing acquaintance with Java programming and the ability to read and understand the Java API specification. This is not a large hurdle for the vast majority of professional developers. An example of an element that might require a Java API customization might be an action element (see Figure 4-3) that invokes the *QueryWebService* java class in order to query an online web service to retrieve the current temperature in a city specified by the user earlier in the call flow. Since studio does not have a prebuilt component to access a Simple Object Access Protocol (SOAP) web service, the developer would have to implement this functionality themselves by creating a custom class that implements functionality in the Cisco CVP and standard Java API.

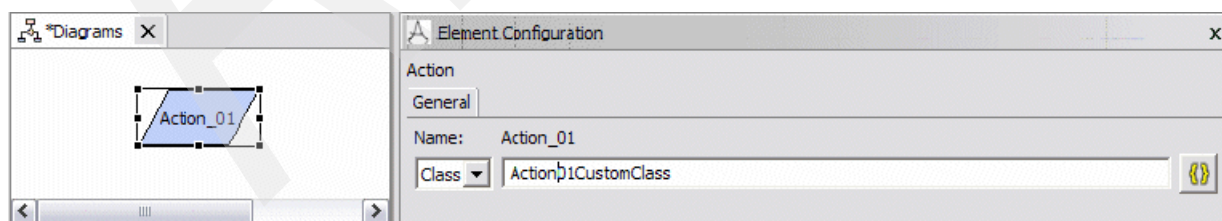


Figure 4-3 Example of an action element whose behavior is defined by the “Action01CustomClass” Java class

## XML-over HTTP Configuration

The XML API allows the developer to use any programming language they wish to in order to implement element customizations. The only requirement is that whatever system is used by

the developer must be able to return an XML document over the HTTP protocol in response to an HTTP request made by the Cisco CVP server. There are several advantages to this approach:

- ▶ The ability to physically isolate business logic and enterprise data from the voice presentation layer of the application by having them run in completely autonomous systems
- ▶ The use of XML, which is common and easy to learn

The only drawbacks to this approach arise from performance issues associated with transmission and processing of HTTP requests between the Cisco CVP server and the responding system.

An example of an element using an XML API configuration (see Figure 4-4) might be a custom audio element that will say Good morning, Good afternoon, or Good evening depending on the time of day the user calls. This would be accomplished by pointing the *audio* setting of the element to the URL of a page that will return a different XML string depending on the time of day specified in the *time* variable of the query string:

```
http://10.10.9.247/getAudioText.php?time={GeneralDateTime.HourOfDay.CURRENT}
```

So, if Cisco CVP requested the following page:

```
http://10.10.9.247/getAudioText.php?time=09:00
```

Then, the XML for Good morning would be returned and the user would hear Good morning.

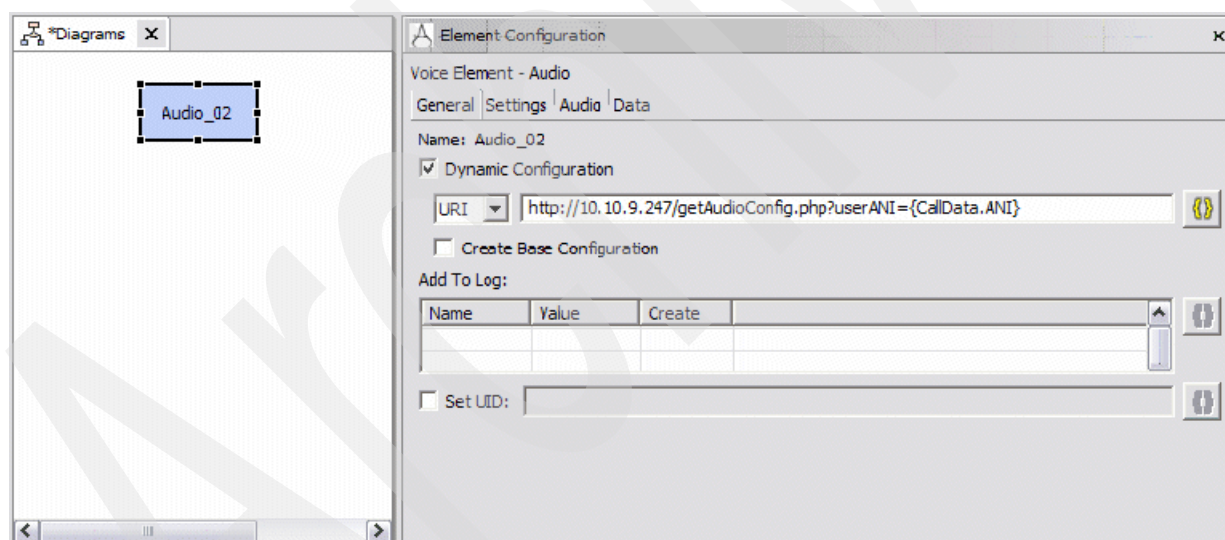


Figure 4-4 Example of an audio element using the XML-over-HTTP API for dynamic configuration

The getAudioConfig.php page will return an XML document (based on the input provided in the userANI variable in the query string) that conforms to the DTD for the Cisco CVP XML API that will tell the audio element how it needs to configure itself.

#### 4.3.4 Overview of Elements

Before going any farther it is important to note that there are five different kinds of elements defined for the Cisco CVP Studio development environment. These are:

- ▶ Voice elements
- ▶ Decision elements

- ▶ VoiceXML insert elements
- ▶ Action elements
- ▶ Flag elements

In addition to these five basic element types the following concepts are also defined:

- ▶ Hot links
- ▶ Hot events
- ▶ Application transfers

Combined these five element types and three concepts are sufficient to fully encapsulate the logic of all voice applications. We will discuss each of these elements in further detail shortly. However before doing so it is important for us to define some key (and sometimes confusing) terminology.

## Standard versus Configurable Elements

Configurable elements come in three different varieties:

- ▶ Voice elements
- ▶ Action elements
- ▶ Decision elements

These elements each have configuration types that are split up into four distinct categories as shown in Table 4-1.

Table 4-1 Element configuration categories

Configuration Type	Description	Decision Elements	Voice Elements	Action Elements
Settings	Used to store information that affects how the element performs. An example of this might be: length of audio input, path to a pre-recorded prompt, etc.	X	X	X
VoiceXML Properties	Equivalent to properties as outlined in the VoiceXML specification. Used to modify element behavior by direct insertion of data into the VoiceXML code that the element produces.		X	
Audio Groups	Nearly all voice elements involve the use of prerecorded audio files or text-to-speech phrases. An audio group encapsulates the audio that will be played to the user when the element is encountered in the call flow.		X	
Variables	This configuration type specifies how the element will access and use element or session data.	X	X	X

Of the configurable elements, voice elements are the most versatile (or complex as the case may be) and include all four types of configuration. Configurable decision and action elements only use the settings and variables types of configuration. In essence, you can think of a configurable element as a prebuilt element the configuration of which a developer has exposed that can be manipulated by the application designer employing that component in order to control how that element will behave within the application. A standard element, on the other hand, is a highly specialized, custom component designed to perform a very specific task within a given application (usually only once, maybe twice). This nomenclature is somewhat counterintuitive, so be careful. An example of a configurable element would be a configurable audio element where we can set an audio group to define what prompt will be read to the user when that element is reached in any given call flow. An extremely trivial example of a standard element would be a *CallRichardsMomAndTellHimHeIsHome* element

that would initiate a new call to Richard’s Mom and tell him that he has arrived at home when a caller named Richard presses the pound key in the *HiMom* application. This is not a terribly reusable component.

We now briefly examine some of the characteristics of various elements that can be found in the studio environment. We engage in a more detailed discussion of many of these elements when we delve into the development of our sample applications.

## Action Elements

Action elements encapsulate business logic and perform tasks that do not affect the call flow. They are elements that only have one exit state (done). They perform whatever back-end processing is required and then immediately pass control on to the next element in the call flow. The action or processing performed by an action element may be specified in a compiled Java class or in a URI that points to a page that will return an XML document compliant with the XML-Over-HTTP API. In essence, an action element is simply a means of inserting some custom code into the a call flow of a voice application. An action element is rendered as a parallelogram in the call flow builder pane of studio as shown in Figure 4-5.



Figure 4-5 Action element configuration

**Note:** Configurable action elements only include settings and variables for their configuration. An action element is rendered as a parallelogram in the call flow editor.

## Decision Elements

Decision elements encapsulate business logic that makes decisions with at least two exit states. A decision element is a branch point in a call flow and provides the necessary processing required to determine where the flow of execution should be directed in a call flow based on specified conditions. Although the vast majority of decisions are of a Boolean nature, as many exit states as are desired can be defined for any given decision. A decision element is rendered as a diamond in the call flow pane of studio as shown in Figure 4-6.

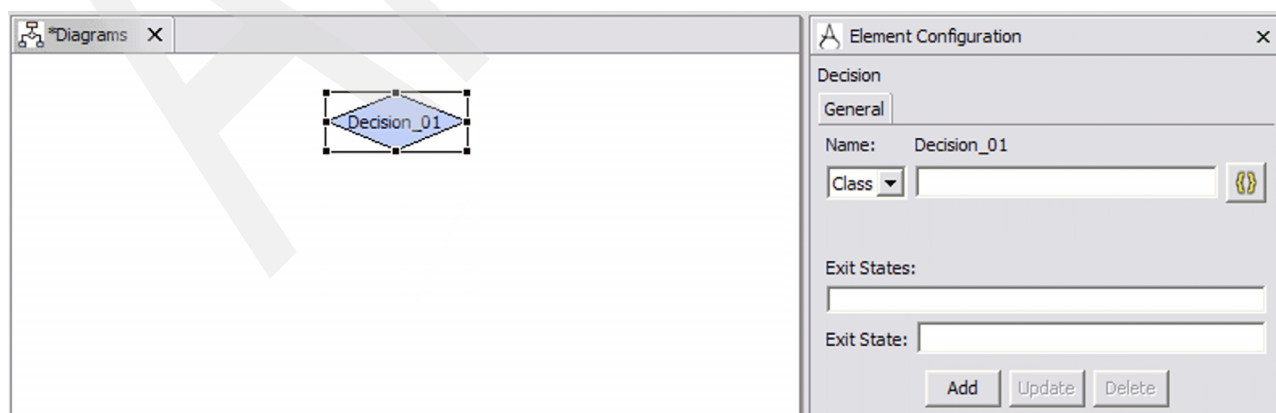


Figure 4-6 Decision element configuration



**Note:** Configurable decision elements only include settings and variables for their configuration. A decision element is rendered as a diamond in the call flow editor.

For example we can have a decision that checks the users Automatic Number Identification (ANI) to see if the user is calling from within a given area code (see Figure 4-7). If so, then the decision element exits with the *within\_area\_code* exit state and execution branches off to the portion of the call flow dedicated to users from within the specified area code. Otherwise, the decision will exit with the *not\_within\_area\_code* exit state and execution will branch off into code to process external callers.

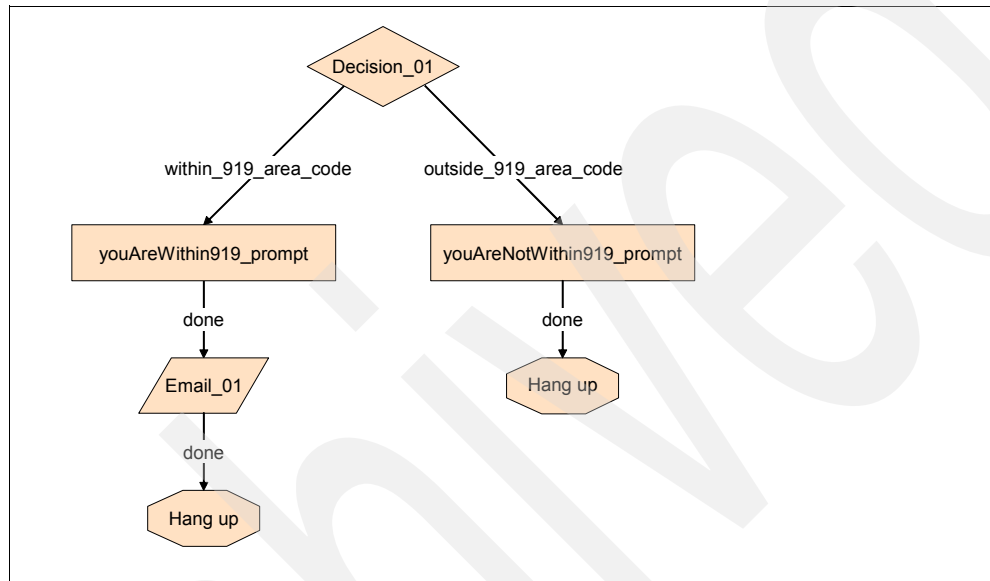


Figure 4-7 An example of a call flow containing a decision element “Decision\_01” with two possible exit states.

## Voice Elements

A voice element is a reusable VoiceXML producing dialog with a fixed or dynamically produced configuration. Voice elements are used to assemble the VoiceXML code sent to the voice browser. Each voice element constitutes a discrete section of a given call and includes several very useful pre-built elements that can be used and reused at various points in any given application (see Table 4-2 on page 111). To configure a voice element fully the developer must provide information for all four types of configuration. See Figure 4-8 on page 113 for an example of a voice element in a call flow.

Table 4-2 Voice element descriptions

Voice Element Name	Brief Description
Audio	Plays any combination of audio files, phrases, or both to be read by the text-to-speech engine
Currency	Captures a currency amount entered by the caller. Currently only supports dollars and cents
Currency_With_Confirm	Same as above, then presents a confirmation menu where the user may re-enter the amount if they desire
Date	Captures a date value entered by the caller

<b>Voice Element Name</b>	<b>Brief Description</b>
Date_With_Confirm	Same as above, then presents a confirmation menu where the user may re-enter the date if they desire
Time	Captures a time value entered by the caller
Time_With_Confirm	Same as above, then presents a confirmation menu where the user may re-enter the time if they desire
Form	Collects input from the user as defined in an inline or external DTMF or speech grammar(s)
Form_With_Confirm	Same as above, then presents a confirmation menu where the user may re-enter the input if they desire
Yes_No_Menu	Presents as a yes/no menu. User input can be defined in an inline or external speech grammar(s). DTMF button 1 is automatically mapped to 'yes' and button 2 is mapped to 'no'
2_ - 10_Option_Menu	Presents a menu to caller with 2 – 10 possible options defined in an inline or external DTMF or speech grammar(s)
Digits	Captures a number (which may include a decimal) entered digit-by-digit by the caller
Digits_With_Confirm	Same as above, then presents a confirmation menu where the user may re-enter the digits if they desire
Number	Captures a natural number (may include a decimal) as entered by the caller
Number_With_Confirm	Same as above, then presents a confirmation menu where the user may re-enter the number if they desire
Phone	Captures a phone number entered by the caller
Phone_With_Confirm	Same as above, then presents a confirmation menu where the user may re-enter the phone number if they desire
Record	Makes a recording of the users voice when prompted. The recording is then saved to the location specified in the settings in the format specified in the settings.
Record_With_Confirm	Same as above, then presents a confirmation menu where the user may re-enter the recording if they desire

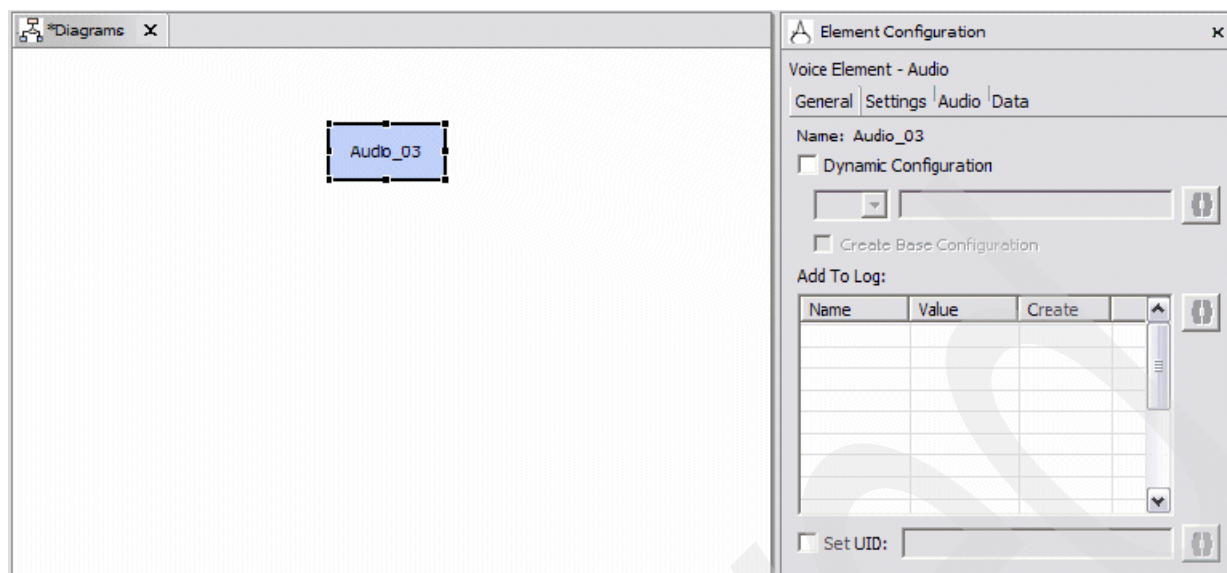


Figure 4-8 Voice Element - Audio configuration

**Note:** Configurable voice elements comprise all four types of configurations (VoiceXML properties, settings, audio groups and variables). A voice element (e.g. audio, form, form\_with\_confirm, etc.) is rendered as a rectangle in the call flow editor.

## Flag Elements

A flag element records when a user has reached a certain point in a given call flow. This provides a mechanism by which an application designer can determine how often a specified point in an application is reached and can be very helpful in terms of determining which portions of an application are more or less popular, problematic, easy or difficult to find. Flag elements are in essence beacons that are triggered once a user visits the portion of the call flow where they reside. An entry in the application log is created every time a flag is tripped and the state of the flag is also stored within the applications call data. Flag elements have a single exit state (done) and have no effect on the flow of execution. A flag element is rendered as a sideways trapezoid in the call flow pane of studio.

## VoiceXML Insert Elements

A VoiceXML-insert is an element built in VoiceXML that provides direct control of lower-level voice dialog at the price of decreased flexibility. There are certain situations when experienced developers might want to include prewritten VoiceXML code in their Cisco CVP voice application in order to obtain finer-grained control over a specific voice function without going to the trouble of creating a full-fledged customized component with the Java API. This is generally a less desirable course of action because certain penalties (such as loss of ability to seamlessly switch between different voice browsers, greater processing overhead, and so on) are incurred with the introduction of raw VoiceXML code into a Cisco CVP application. A VoiceXML-insert element can have as many exit states as the developer wants with a minimum number of one.

## Hotevents

A HotEvent is a global event (may be *user-triggered* such as a *noInput* event, *asynchronous* such as an *error.badfetch* thrown by the voice browser, or *developer defined* such as a hotlink

that throws an event) that when caught executes developer-specified actions. A hotevent is rendered as an upside down trapezoid in the call flow pane of studio.

## Hotlinks

Hotlinks are globally accessible utterances and key presses that immediately bring the call to a specific part if the call flow throws an event. One common example of an implementation that would require a hotlink is if we want the user to be able to say the word Operator or press the 0 key at any point in the application to be redirected to a live representative. A hotlink can be activated by a spoken keyword, a DTMF entry, or both. A hotlink has a single exit state that must be connected to the point in the call flow that the developer wishes the flow of execution to be diverted to when the hotlink is activated. The developer can also specify an event that should be thrown when the hotlink is activated by the appropriate utterance or key press.

## Application Transfers

An application transfer is a transfer from one voice application to another running on the same VoiceXML server system, simulating a new phone call. While most CVP voice applications are expected to function as standalone units there is always the possibility that a caller will need to be transferred between applications running on the same system. An application transfer is not a true telephony transfer (hook flash) per se, but rather a server-side software emulation of the same. For logging purposes, the server will treat the call into the new application as an entirely new call. An application transfer has no defined exit states. Element and session data can be passed between applications. An application transfer is rendered as an oval in the call flow pane of studio.

## Page Connectors and Page Entries

Generally speaking larger and more complex applications will have call flow diagrams that do not readily fit into the space on one page in the call flow editor. Page Entries and Page Connectors are used to connect call flows that span multiple pages. A Page Entry is the entrance point to a specific part of the call flow. A Page Connector is the means by which the call flow moves to a specific entry. A Page Entry is rendered as an inverse arrow and a Page Connector is rendered as an arrow in the call flow pane of studio.

Figure 4-9 on page 115 and Figure 4-10 on page 116 illustrate a trivial call flow that spans two pages and is connected by means of Page Entry ("page\_1" and "page\_2") and Page Connector elements.

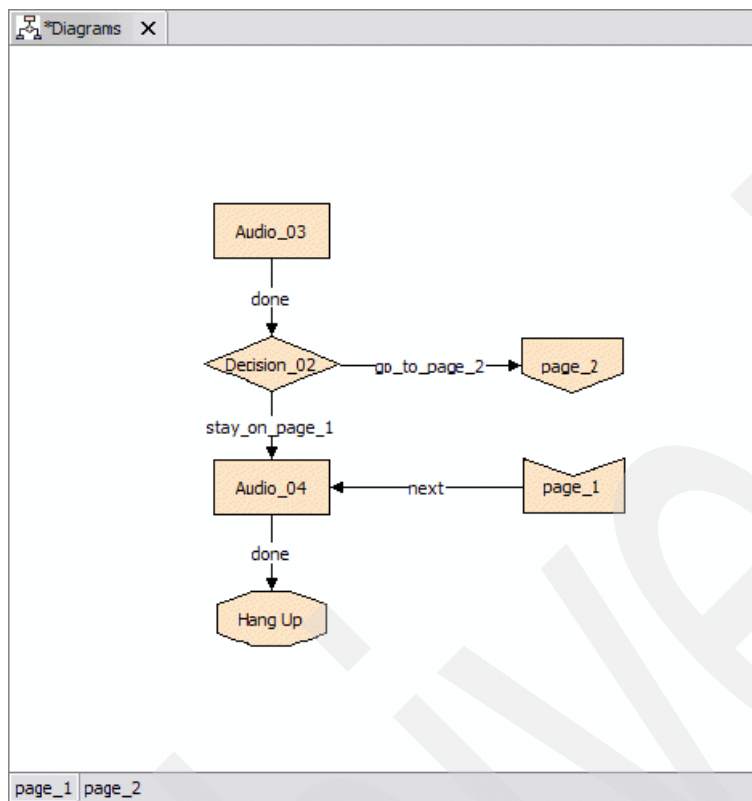


Figure 4-9 Page 1 of the call flow

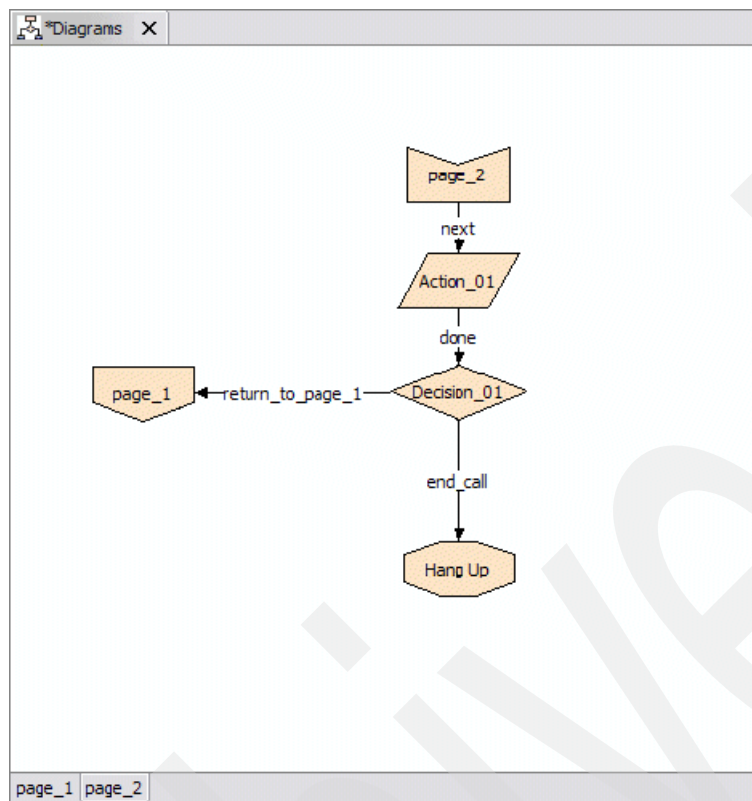


Figure 4-10 Page 2 of the call flow

## Comments

Comments are notes added to a call flow by the developer for informational purposes. These can be very handy to make call flows more understandable and for documentation purposes when working in a collaborative environment. Developers are encouraged to use comments freely throughout their call flows. To define a comment, simply right-click the comment and choose **Edit Comment**. A dialog box opens in which the comment can be typed. Figure 4-11 shows an example of a comment in a call flow.

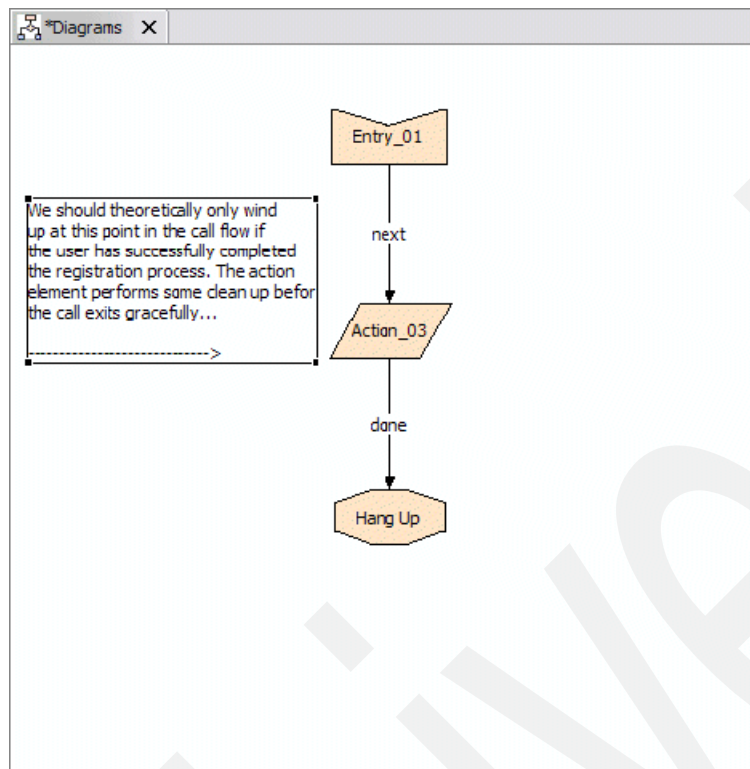


Figure 4-11 Comments make call flows far more intelligible and should be used to clarify whenever it is deemed necessary.

## Hang Ups

A Hang Up is a location in the call flow where the exits the call and hangs up on the caller. A voice application can have as many hang ups as the developer wants. The Hang Up element is rendered as an octagon in the call flow pane of studio.

## Start of Call

The Start of Call element is a special page entry that points to the component that is to be visited at the start of every call. By default, every Cisco CVP project is created with a single page containing a start of call element.

## Error Elements

An Error Element is a voice element whose purpose is to provide appropriate handling for the call when an unexpected error occurs. The error element does not actually deal with the error on an application level, but rather handles the caller. That is to say when an unexpected error occurs, an error element will allow for the caller to be transferred to a live agent or perhaps to play an error message, or any other type of caller handling that would qualify as a graceful exit from the application where the unexpected error occurred. Error elements are designed to deal with exceptional conditions not originally envisioned by the developer, such as *error.badfetch* raised as a result of a missing external grammar file or an exception thrown by a custom Java class, for example. An error element can be created from any configurable voice element by simply right-clicking on the element and selecting **Error Element** → **Yes**. The shape of the element in question will then change to a triangle. It is important to note that there can only be one error element defined for a particular application.

## Cisco Proprietary Elements

The following are Cisco Proprietary elements:

- ▶ subdialog\_start
- ▶ subdialog\_return

## 4.4 Installing Studio and Server

There are several possible installation options available to the developer. Generally we recommend that the developer perform a full installation of the Cisco CVP Studio and Server so that applications can be developed and deployed locally. This simplifies the deployment process somewhat and makes the testing process somewhat easier. However for various reasons (such as limited availability of licenses) you may wish to install only the Cisco CVP VoiceXML Studio software on your desktop and deploy compiled applications to a remote server. We will detail all possible installation options to ensure that you have a full understanding.

### 4.4.1 Full Installation

This section describes the process of performing a full installation of Cisco CVP VoiceXML consisting of CVP VoiceXML Server, CVP VoiceXML Studio, Apache Tomcat V4.1.24 and Java V1.4.2 J2SE SDK.

#### Prerequisites

Before you begin, be certain these conditions are met:

- ▶ All the requirements described in the section entitled System Requirements are met for both Studio and Server.
- ▶ The software should be installed by the user that is expected to run Cisco CVP VoiceXML Server. For example, if the Server is expected to run under the user Administrator, the installer must be run under Administrator as well. This is especially important when the application server is installed as a service.

#### Installation

Follow these steps to complete the installation.

1. Run the Cisco CVP VoiceXML installer by executing **CiscoCVP.exe**. The installer will begin extracting the files as shown in Figure 4-12.

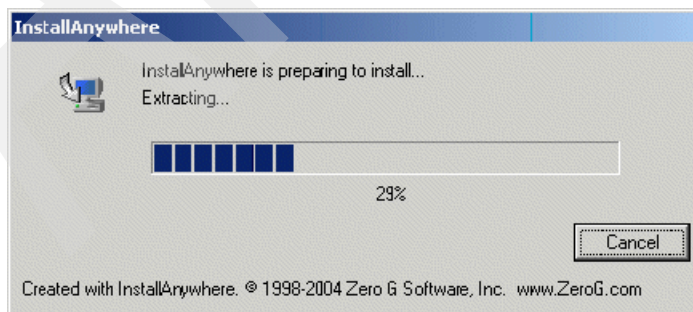


Figure 4-12 Extracting the installation files

Then you see an Introduction dialog box as shown in Figure 4-13.



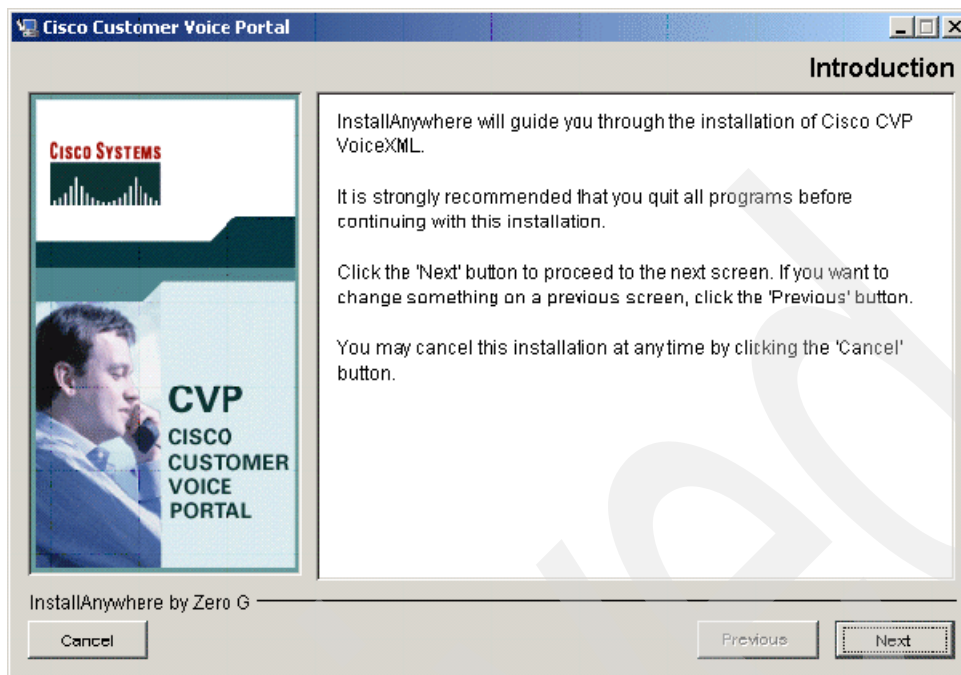


Figure 4-13 Introduction

2. Click **Next** and you are presented with a License Agreement dialog box as shown in Figure 4-14 on page 119.

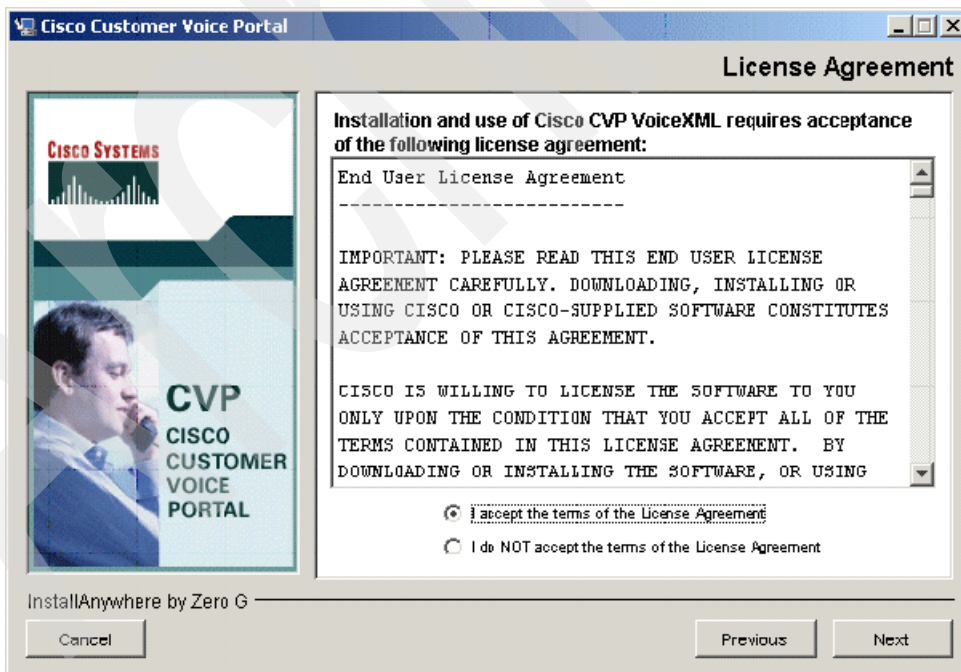


Figure 4-14 License Agreement

3. Select **I accept the terms of the License Agreement** and then click **Next**. The installer asks for the type of the installation to choose. Select **Full** and click **Next** as shown in Figure 4-15 on page 120.

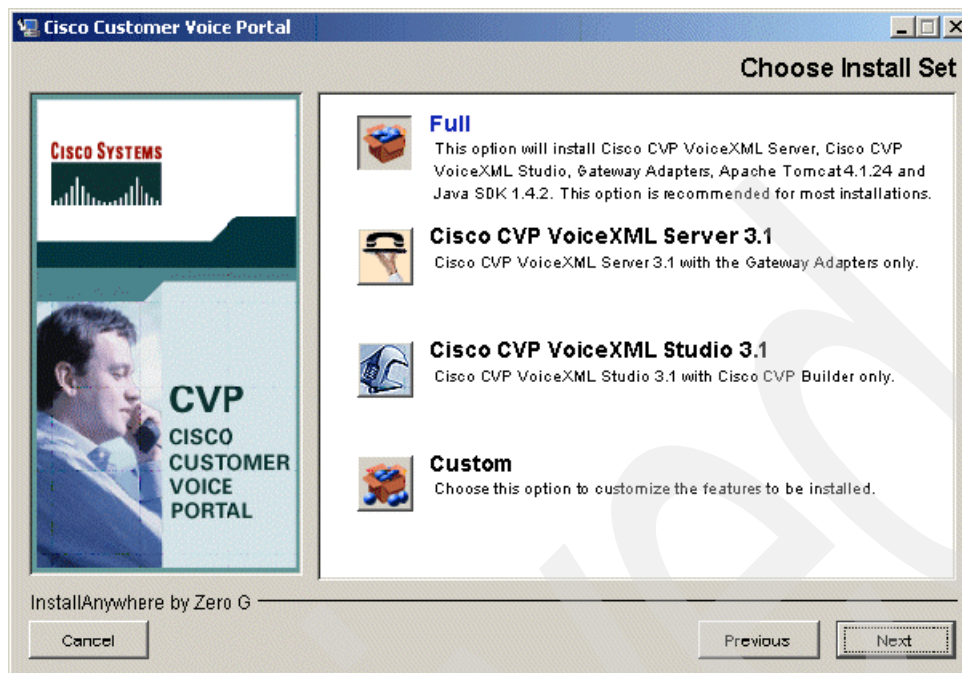


Figure 4-15 Choose Install Set

4. Choose whether to create desktop and program menu shortcuts for CVP VoiceXML Studio and click **Next**.
5. Fill in the Administrator login information for the Tomcat V4.1.24 installation. If the default values are used, there will be no administrator password and the HTTP/1.1 Connector port will be 8080. Click **Next** to continue with the installation as shown in Figure 4-16 on page 120.

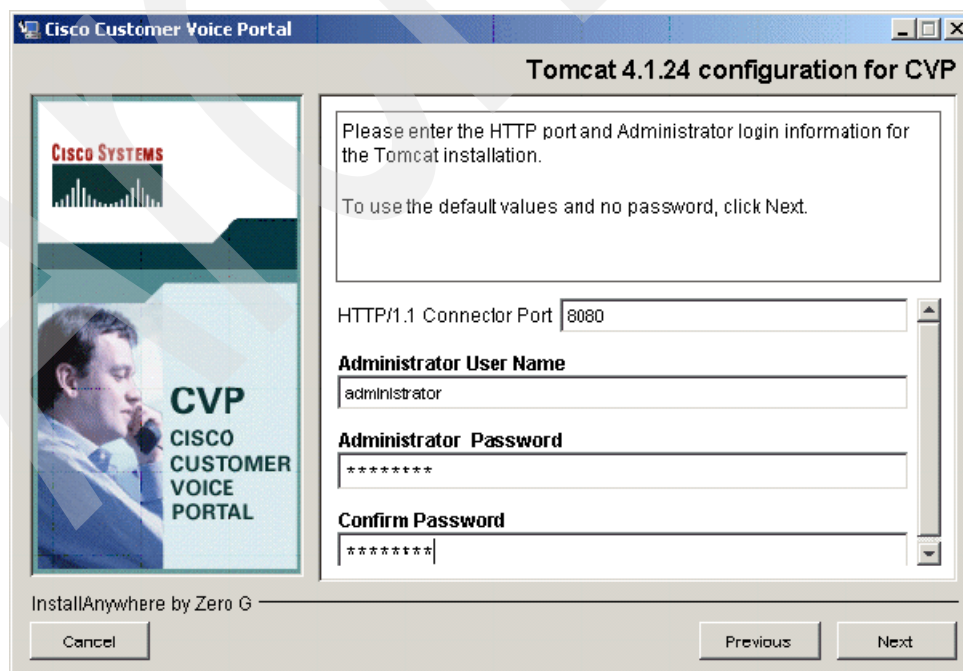


Figure 4-16 Tomcat V4.1.24 configuration for Cisco CVP

6. Decide whether to install Tomcat V4.1.24 as a Windows service as shown in Figure 4-17 on page 121. If Tomcat is installed as a service, Cisco CVP VoiceXML Server will be started automatically upon Windows startup. Click **Next** to continue.

**Note:** On a development machine you might want to answer **No** to this question. As you will see later, it can be very helpful to start the server manually from the command line for the purposes of redirecting standard output to a text file for debugging.

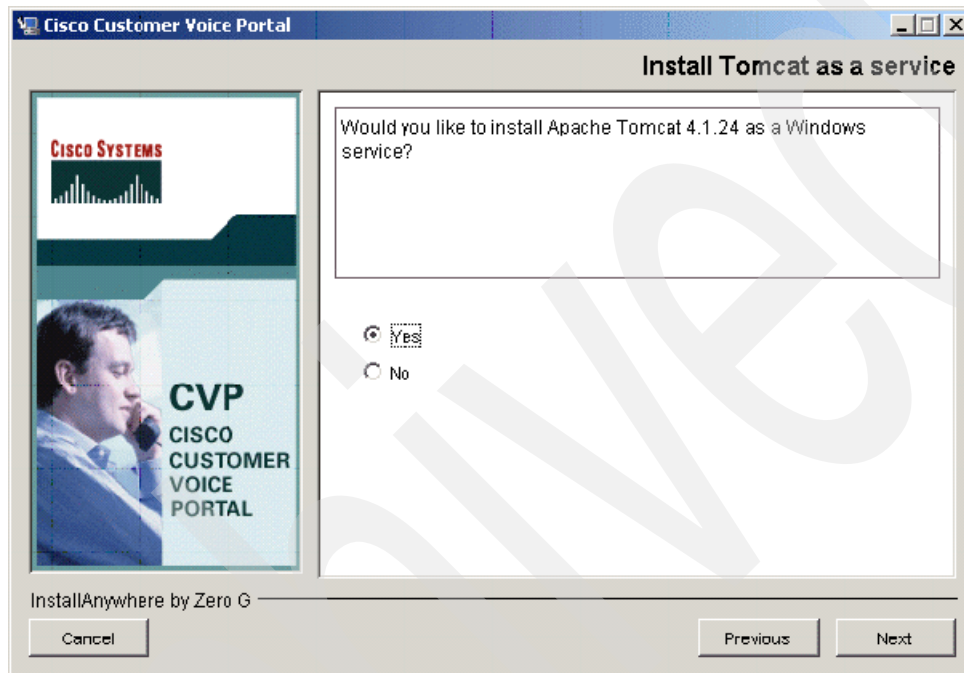


Figure 4-17 Install Tomcat as a service

7. Figure 4-18 on page 122 summarizes the chosen options and installation path. Review the Pre-Installation Summary carefully before selecting **Install**. After the selection, the installation of Cisco CVP VoiceXML begins.

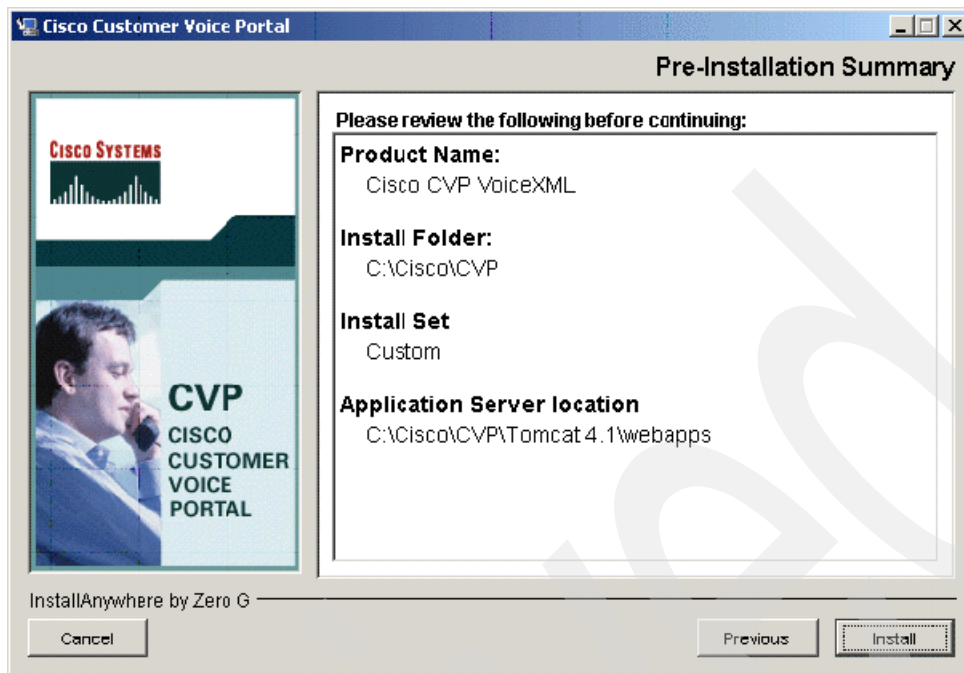


Figure 4-18 Pre-Installation Summary

8. Cisco CVP VoiceXML Server requires the use of an environment variable that is created by the installer. In order for this environment variable to take effect, the machine must be rebooted. After a successful installation, click **Next** and you are asked to restart your computer as shown in Figure 4-19 on page 122.

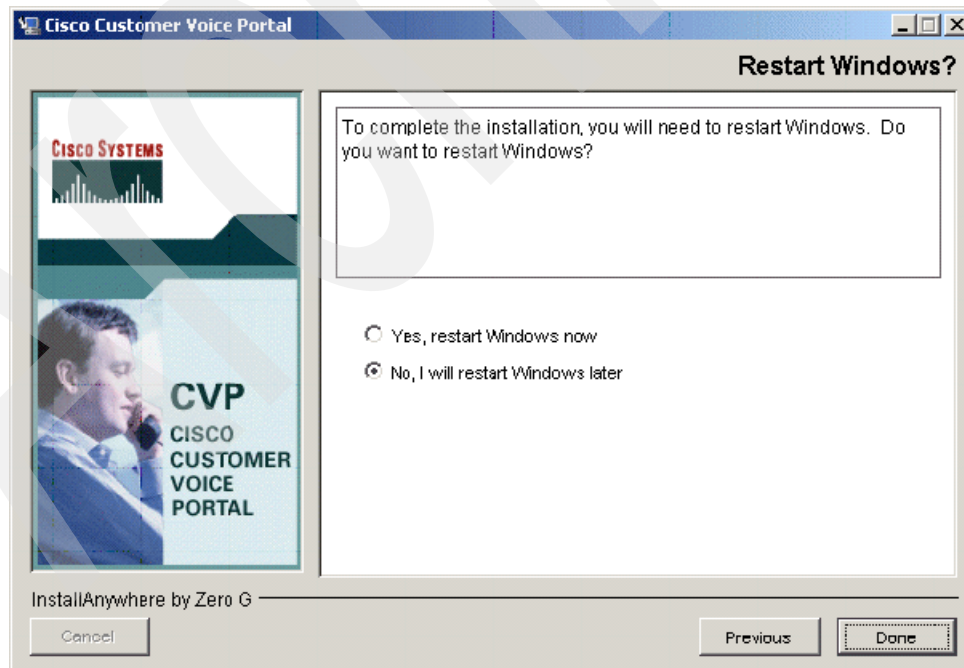


Figure 4-19 Restart Windows

9. Click **Done** to complete the installation.

You now have to place the license files in the appropriate directories in order for the server and studio software to launch successfully.

#### 4.4.2 Cisco CVP VoiceXML Server V3.1 Only Installation

This section describes the process of installing only Cisco CVP VoiceXML Server.

##### Prerequisites

Be certain the following conditions are met:

- ▶ All the requirements described in the section entitled System Requirements are met for Cisco CVP VoiceXML Server.
- ▶ The software should be installed by the user that is expected to run Cisco CVP VoiceXML Server. For example, if the Server is expected to run under the user Administrator, the installer must be run under Administrator as well. This is especially important when the application server is installed as a service.

##### Installation

Follow these steps to complete the installation.

1. Run the Cisco CVP VoiceXML installer by executing **CiscoCVP.exe**.
2. After agreeing to the license agreement, the installer asks for the type of the installation to choose. Select **Cisco CVP VoiceXML Server V3.1** and click **Next**.
3. You are asked to provide the installation folder for Cisco CVP VoiceXML in which the Server will be installed. The default location is C:\Cisco\CVP. The installation directory can be changed by manually editing the path or by clicking **Choose** to choose another location. This directory will be referred to in this guide as *INSTALLATION\_PATH*.

**Note:** If Cisco CVP VoiceXML has already been installed, the existing installation path is used automatically and this step is skipped.

4. Select the application server on which to install Cisco CVP VoiceXML Server. The options for application server are:
  - Apache Tomcat V4.1.24
  - WebSphere V5.1
  - Other
5. The installer asks for the location of the deployable Web applications folder of your application server in order to install the Server. The installer tries to obtain this path from the operating system or failing that, set it to the default installation directory for the selected application server. The installation directory can be changed by manually editing the path or by clicking **Choose** to set another location.

**Note:** If you chose **Other** in step 4, the CVP.war file will be installed to C:\Cisco\CVP\War (for manual deployment) or you can enter the path where Web applications are deployed for the appropriate application server.

6. The next dialog box summarizes the chosen options and installation path. Review your selection carefully before selecting **Install**. The installer will then install Cisco CVP VoiceXML Server.
7. Cisco CVP VoiceXML Server requires the use of an environment variable that is created by the installer. In order for this environment variable to take effect, the machine must be

rebooted. After a successful installation, click **Next** and you are asked to restart your computer.

8. Click **Done** to complete the installation.
9. Refer to the Post Installation section for instructions on licensing and verifying the installation.

**Note:** If WebSphere Application Server V5.1 has been selected for the application server, refer to the Continuing CVP Server Installation on WebSphere Application Server V5.1 section.

### 4.4.3 Cisco CVP VoiceXML Studio V3.1 Only Installation

This section describes the process of installing only Cisco CVP VoiceXML Studio.

#### Prerequisites

Cisco CVP VoiceXML Studio must be installed on Microsoft Windows 2000/XP operating system.

#### Installation

Follow these steps to complete the installation.

1. Launch the Cisco CVP VoiceXML installer with the filename **CiscoCVP.exe**.
2. After agreeing to the license agreement, the installer asks for the type of the installation to choose. Select **Cisco CVP VoiceXML Studio V3.1**.
3. You are asked to provide the installation folder for Cisco CVP VoiceXML (in which Studio will be installed). The default location is C:\Cisco\CVP. The installation directory can be changed by manually editing the path or clicking **Choose** to select another location. This directory will be referred to in this guide as `INSTALLATION_PATH`.

**Note:** If Cisco CVP VoiceXML has already been installed, the existing installation path is used automatically and this step is skipped.

4. After choosing whether to create shortcuts to Cisco CVP VoiceXML Studio, the next screen summarizes the chosen options and installation path. Review your selection carefully before selecting **Install**. The installer will then install Studio.
5. Click **Done** to complete the installation.
6. Refer to the Post Installation section for instructions on licensing and verifying the installation.

## 4.5 First steps in VoiceXML development in studio

In this section we take our first steps into the world of voice application development in the CVP VoiceXML suite. We will start by developing the classic, trivial application HelloWorld which will simply read a text-to-speech string back to the user that says *Hello world*. As we progress, we gradually ramp up the functionality of our HelloWorld application to include various commonly used elements and design techniques. When we have thoroughly exhausted the possibilities inherent in the venerable HelloWorld, we move on to discuss a more realistic customer-facing self-service application that you might encounter in the real world and touch on the topics of using the Java API to create customized configurable



components, integration within back-end data repositories and other such interesting topics. But first things first, let us look at what we have to do to create our project and initiate the development process.

### 4.5.1 Creating the HelloWorld project

Cisco CVP VoiceXML studio employs a *project* paradigm. This paradigm should be familiar to you if you have developed Java applications in the Eclipse framework, or other commonly used Integrated Development Environments. A studio project contains all the resources required to build and deploy a functional voice application that will run on a Cisco CVP server. The *callflow* folder contains the XML files that define the static configurations for all elements in the application's call flow as well as the application's call flow itself. The *app.callflow* file when opened in studio will appear as a flow chart outlining the business logic of the application as defined in the various XML files. The *deploy* folder is where extra resources (such as Java class files, external libraries, local custom elements, extra Say It Smart plug-ins, etc.) associated with the deployed application are placed.

Figure 4-20 on page 125 shows a preview of the default directory structure of a functional Cisco CVP voice application as seen from the project navigation panel.

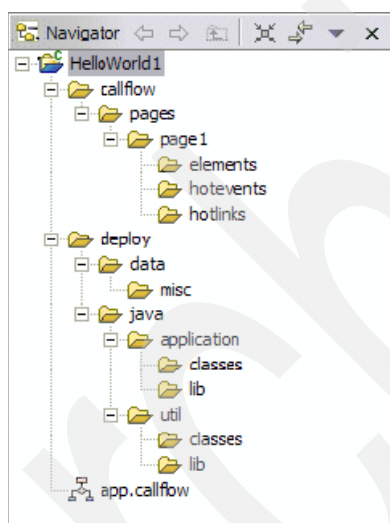


Figure 4-20 Project Navigation panel

### Creating a new Cisco CVP VoiceXML Studio Project

Now, before we actually start developing our application, we first must set up our environment by creating a new project in which to work. This is a relatively simple process that is guided by wizards. Let us quickly go through the steps that are required to create a new studio project.

#### ***Using the wizard to create a new project***

To create a new project, follow these steps:

1. First we launch studio and then navigate to **File** → **New** → **Cisco CVP VoiceXML Studio Project** as shown in Figure 4-21 on page 126.

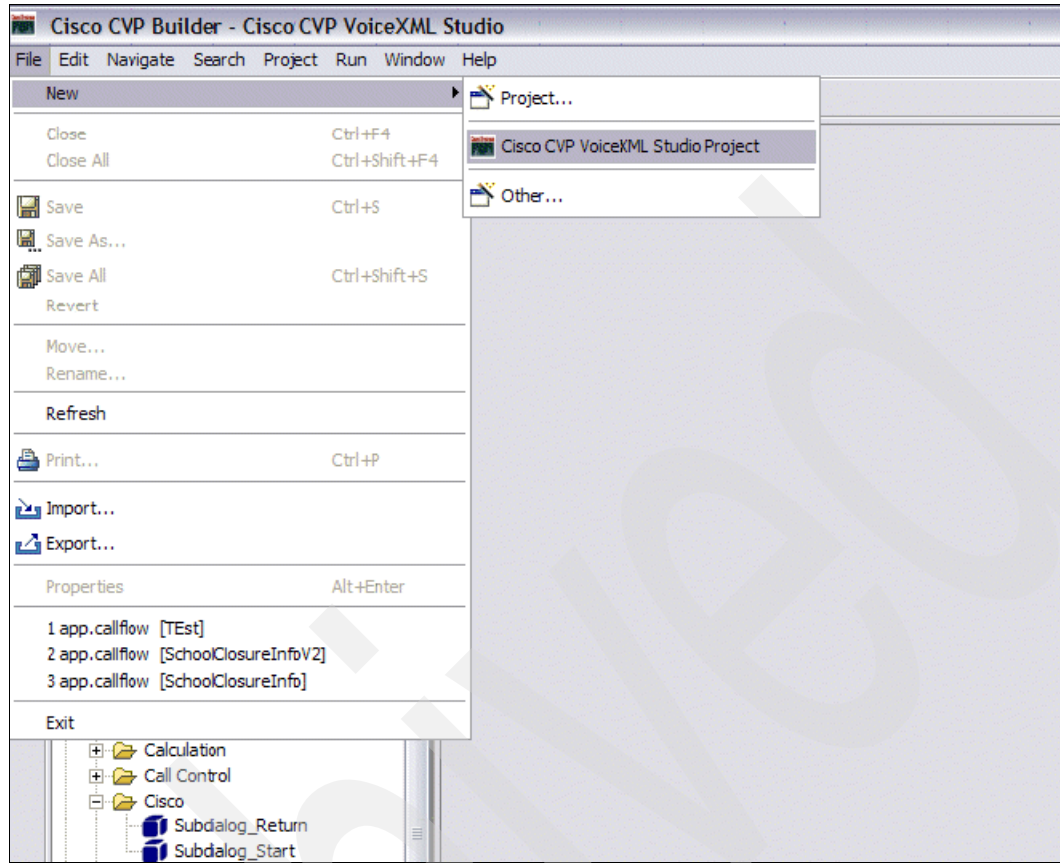


Figure 4-21 Creating a new Cisco VoiceXML Studio Project

Figure 4-22 shows the New Project dialog box.

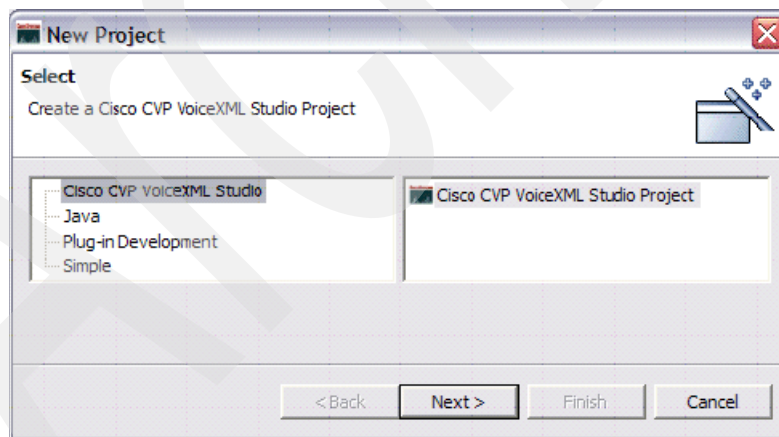


Figure 4-22 New Project - Cisco CVP VoiceXML Studio Project

2. Now we must enter a name for our newly created project (see Figure 4-23). We call this one HelloWorld1. Leave the **Use default** box checked for Project Contents. This will simply place the contents of your project in the default location, which is a perfectly acceptable setting under almost all circumstances.



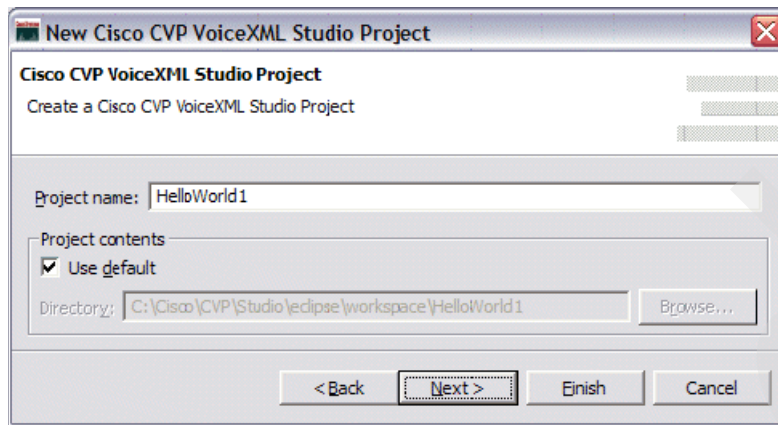


Figure 4-23 New Cisco CVP VoiceXML Studio Project: Project name

3. We now come to a dialog box that specifies the General settings for the project (see Figure 4-24 on page 128). For the vast majority of these settings (particularly for our introductory example) the default settings suffice. The only setting that we modify for our project is the **VoiceXML Gateway** setting, which we change to **Cisco CVP V3.1 with IBM WebSphere Voice Server V5.1**. Selecting this option causes our studio application to generate VoiceXML code that is compliant with the browser on the Cisco gateway when the application is compiled and deployed to the Cisco CVP server.

Following, you can see a more detailed description of the General project settings. This information can come in handy when working on more advanced projects that require modifications to these settings. You can also populate the maintainer field with the e-mail address of the application's primary administrator. If the voice browser encounters any unhandled exceptions while running this application, an e-mail notification is sent to this address. When you have finished with this dialog box, click **Next**.

**New Cisco CVP VoiceXML Studio Project**

**Cisco CVP VoiceXML Studio Project**

General Settings

Maintainer:

Language:

Encoding:

Session Timeout:  minutes

VoiceXML Gateway:

Logging Level:

User Management: ☐ Enable

JNDI Name:

On Call Start:

☐ Run In Background

On Call End:

< Back   Next >   Finish   Cancel

Figure 4-24 New Cisco CVP VoiceXML Studio Project - General Settings

4. We now reach the dialog box where Audio settings for the project are specified (see Figure 4-25 on page 129). In general terms, we specify where various commonly used audio resources for the project can be found. For the purposes of the introductory application, we are not using any prerecorded audio, so we simply leave all properties at their default settings and click **Next**. Later, we will present a more detailed explanation of the audio properties that you can modify in development of more complex projects.

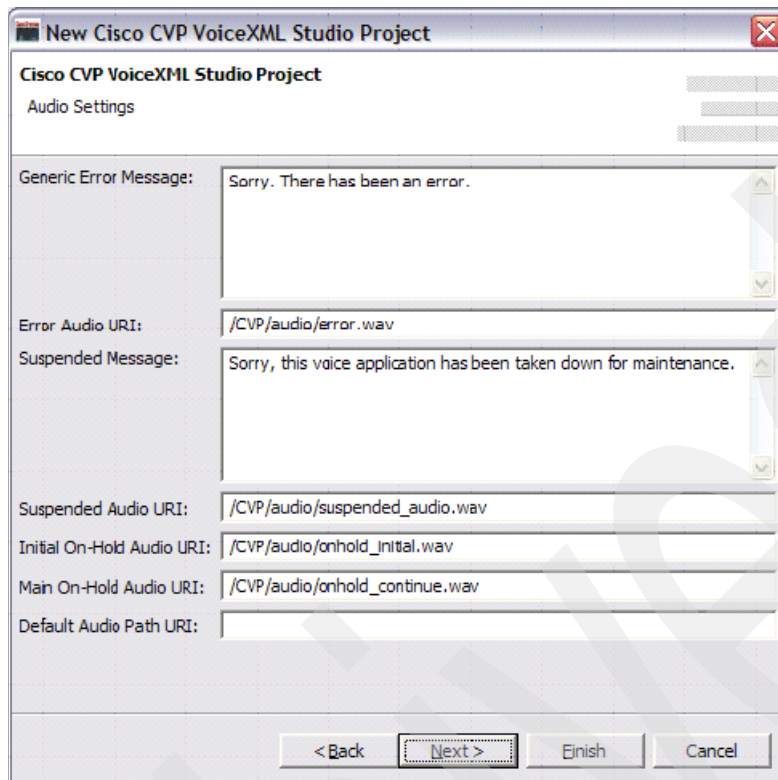


Figure 4-25 New Cisco CVP VoiceXML Studio Project - Audio Settings

- Finally, we reach the screen where Root Document Settings can be specified for our new project (see Figure 4-26 on page 129). For the purposes of our introductory application, we will leave these settings blank. A more detailed explanation of these properties will be presented later. For now, we will simply click **Finish** and start designing our trivial application.

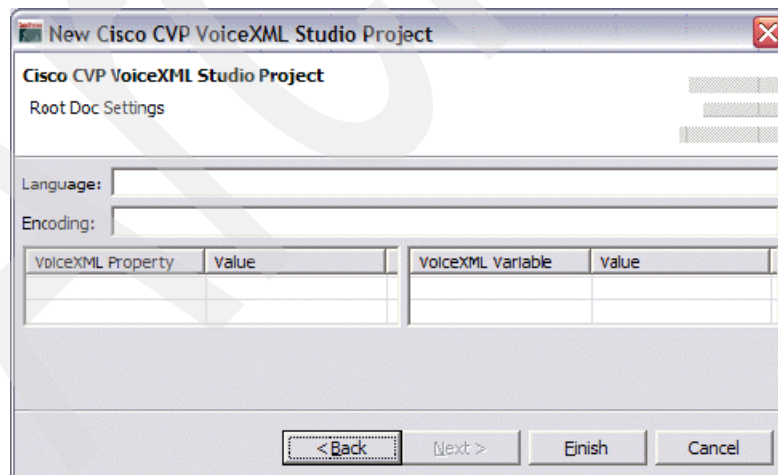


Figure 4-26 New Cisco CVP VoiceXML Studio Project - Root Doc Settings

## General Settings

The following are detailed descriptions for the General Settings.

### ***Maintainer***

This field contains the e-mail address of the application's administrator. If the voice browser on the gateway encounters an unhandled exception it will send an e-mail notification to the address specified here. This can be a very convenient way for administrators to keep track of errors that the application developer did not anticipate that may be cropping up while an application is running in a production environment. It is generally a good idea to populate this field.

### ***Language***

This contains the language to specify in each VoiceXML documents "xml:lang" attribute, for example en-US would be used to specify that the application's language is US English and ca-FR to specify Canadian French. This can come in very handy when creating applications in a multilingual environment. For a more detailed explanation of the 'xml:lang' attribute, see the following URLs for RFC1766:

<http://www.ietf.org/rfc/rfc1766.txt>  
<http://w3.org/TR/2000/REC-xml-20001006#RFC1766>

### ***Encoding***

This field contains the encoding that will be used to create the VoiceXML document. By default this field is blank representing the server default encoding. It is generally a good idea to leave this setting at the default value unless you have a specific reason for overriding it.

### ***Session Timeout***

This value specifies the length of time the application session will wait before expiring a browser session. It is generally a good idea to leave this value at the default value of 30 minutes, unless you are an experienced developer with a specific justification for wanting to extend or shorten the length of a browser session. Thirty minutes is generally just long enough that we can be sure that sessions will not die prematurely and short enough that we will not have unused idle sessions using up system resources on the application server.

### ***VoiceXML gateway***

This property specifies the type of voice browser on which the deployed application will be hosted. There can only be one voice browser per application, because the compiled application deployed to the CVP server will generate VoiceXML specific to the browser specified here.

**Troubleshooting Tip:** An interesting fact to note is that improper setting of this property can sometimes lead to bewildering and frustrating errors. If you notice that you have deployed an application that has no apparent errors in it and the application refuses to accept speech input (it seems that ASR attempts are constantly failing) but your application seems to recognize DTMF key presses successfully, then it is very likely that you have accidentally set this property to Cisco CVP V3.1 for DTMF Only when it should be set to **CVP V3.1 with IBM WebSphere Voice Server V5.1** for a speech enabled application.

### ***Logging level***

This property specifies the verbosity of application log entries. There are three possible settings:

- ▶ none
- ▶ moderate
- ▶ complete

Generally, we recommend that log verbosity be set to the highest level during the development process because log entries can serve as an invaluable source of information

for debugging and troubleshooting. This setting is also generally acceptable for deployment of the application to the production environment as logging functions are not particularly resource intensive and the log files will take quite a while to reach unwieldy sizes.

### ***User management***

Clicking the User Management check box activates Cisco CVP's built-in user management system for this application. While a discussion of the User Management system is outside the scope of this Redpaper, we mention that the database type must be specified (MySQL and Microsoft SQL server are currently supported) and the JNDI identifying the database to be used must be specified as well. By default, the user management system is disabled.

### ***On call start***

This specifies the Java class or the URI (for configuration with the XML-over-HTTP API) that is accessed when the application is entered by a new caller. If the **Run In Background** is selected, then the class or URI will run and the call flow commences without waiting for the successful return from the specified code. Otherwise, the execution of the application does not commence until the specified code has successfully finished executing. For a Java class, the full name (including the package name should be specified). For a URI an absolute URI with a fully qualified domain name should be specified.

### ***On call end***

This specifies the Java class or the URI (for configuration via the XML-over-HTTP API) that will execute when the application visit ends. For a Java class the full name (including the package name should be specified) for a URI an absolute URI with a fully qualified domain name should be specified.

## **Audio Settings**

The following are detailed descriptions for the Audio Settings.

### ***Generic Error Message***

This is the TTS string that will be played if the Cisco CVP server encounters an unhandled exception during the course of a call. The idea behind this is to exit from a call gracefully when an unhandled exception is encountered rather than just abruptly disconnecting the user. The system will disconnect the user immediately after playing this message so it should be tailored according to your business needs in terms of courtesy and verbosity. This is a required setting.

### ***Error Audio URI***

Similar to the "Generic Error Message" setting except that this is a URI pointing to a prerecorded audio file. If this property is specified, then the TTS string specified in the Generic Error Message property will be played only if this file cannot be found or is corrupted. By default, the URI for a Cisco provided audio file is specified. You may override this to point to an audio file of your choosing.

### ***Suspended Message***

This is a TTS message that will be played to callers when the application is suspended (i.e. temporarily down).

### ***Suspended Audio URI***

Similar to the Suspended Message property, this URI points to a prerecorded audio file that will be played to the caller when the application is suspended.

### ***Initial and Main On-Hold Audio URI***

If a call is received by the server when all licensed ports are occupied (each simultaneous call to the system will occupy a single port) the caller will be placed on hold until a port becomes

available. The audio file pointed to by this URI will be played once at the beginning of the hold period and will then proceed to repeatedly play the file referenced by the Main on-Hold Audio URI property until a port becomes free and the user is allowed to interact with the application in question.

### ***Default Audio Path URI***

This is a particularly important setting. This property contains a partial path to the directory that contains prerecorded audio content for this application. If this property is set then the developer will only have to provide the file name when specifying audio resource paths in the studio environment.

**Design Tip:** It is generally considered a good design practice to try to consolidate all audio resources for an application under a single root directory so that they are readily located. If it is feasible, this directory should be on the Cisco CVP server itself. If you have installed the server on a Windows machine, then a good location for all audio files could be the system path:

```
C:\Cisco\CVP\Server\Tomcat 4.1\Webapps\CVP\audio
```

You can then set the default audio path for your application to:

```
http://localhost:8080/CVP/audio
```

Then simply place all audio files used by the application in the following directory on the Cisco CVP server and you should always be able to find your audio files in your deployed application:

```
C:\Cisco\CVP\Server\Tomcat 4.1\Webapps\CVP\audio
```

## **Root document settings**

The following are detailed descriptions for the Root Document Settings.

### ***Language***

This property contains the value for the root document's "xml:lang" attribute. By using this option in the root document, the developer sets the language for the entire application. Leaving this blank represents the use of the application server default setting for the entire application.

### ***Encoding***

This field contains the encoding that will be used to create the VoiceXML document. By default, this field is blank representing the server default encoding. It is generally a good idea to leave this setting at the default value, unless you have a specific reason for overriding it.

### ***VoiceXML Property***

This table is used by the developer to enter any VoiceXML properties that they would like to have appear in the root document for the application. The property names are entered alongside their value. By applying these properties to the root document they will be visible throughout the entire scope of the application.

### ***VoiceXML Variables***

This table is used by the developer to specify any VoiceXML variables that they would like to have appear in the root document for this application. The variable name is entered alongside the value for the variable. By placing these variables in the root document, they will be visible throughout the entire scope of the application.

## Import an existing project into studio

Before proceeding to actual development of our introductory application it is important to note that creation of a brand new project via the new project wizard is not the only way to bring a project to life in studio. It is also possible to import existing projects very easily. Simply follow this procedure:

1. Navigate to **File** → **Import** as shown in Figure 4-27 on page 133.

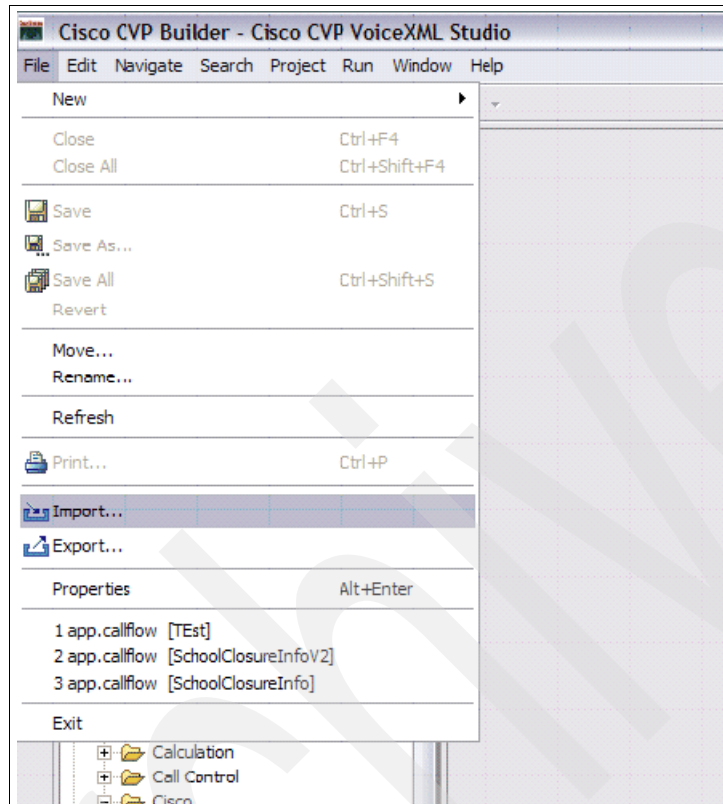


Figure 4-27 Import an existing project

2. Select **Existing Project Into Workspace** and click **Next** as shown in Figure 4-28 on page 134.



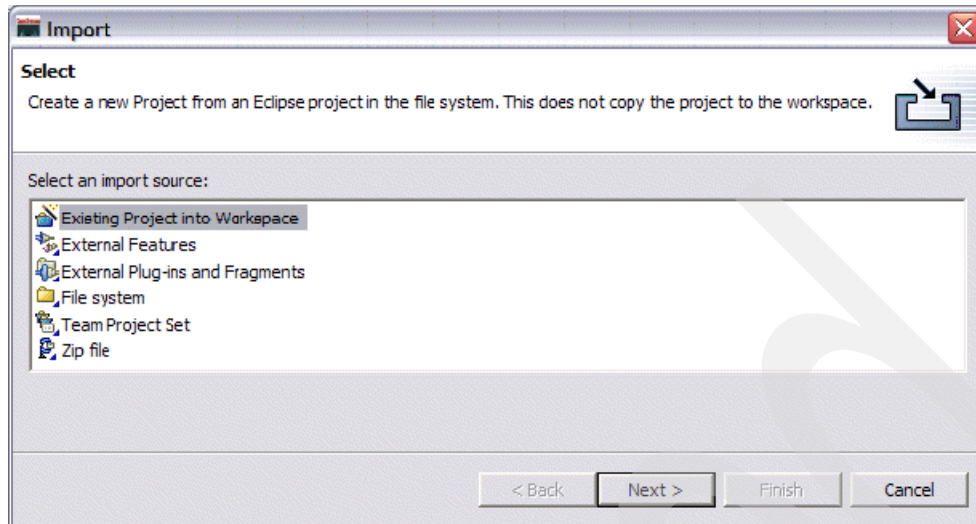


Figure 4-28 Import - Select an import source

3. Enter the location of the existing project that you would like to import or, alternately, click **Browse** to specify the location in the file system (see Figure 4-29).

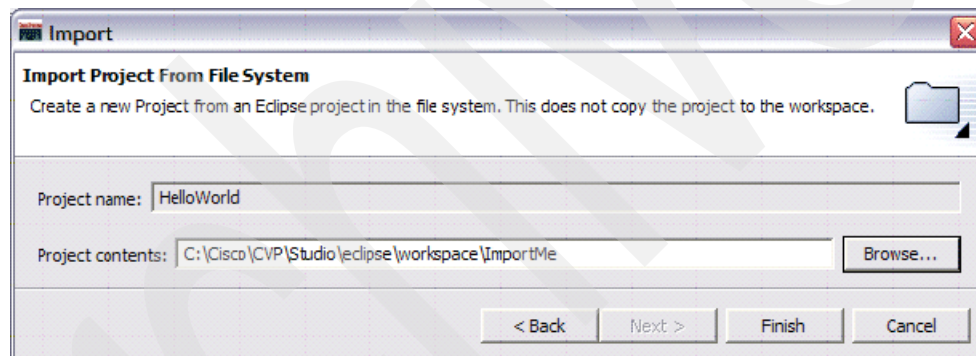


Figure 4-29 Import Project From File System

4. When you have located the project you would like to import, simply click **Finish** and the imported project will now appear in the studio workspace as shown in Figure 4-30 on page 135.



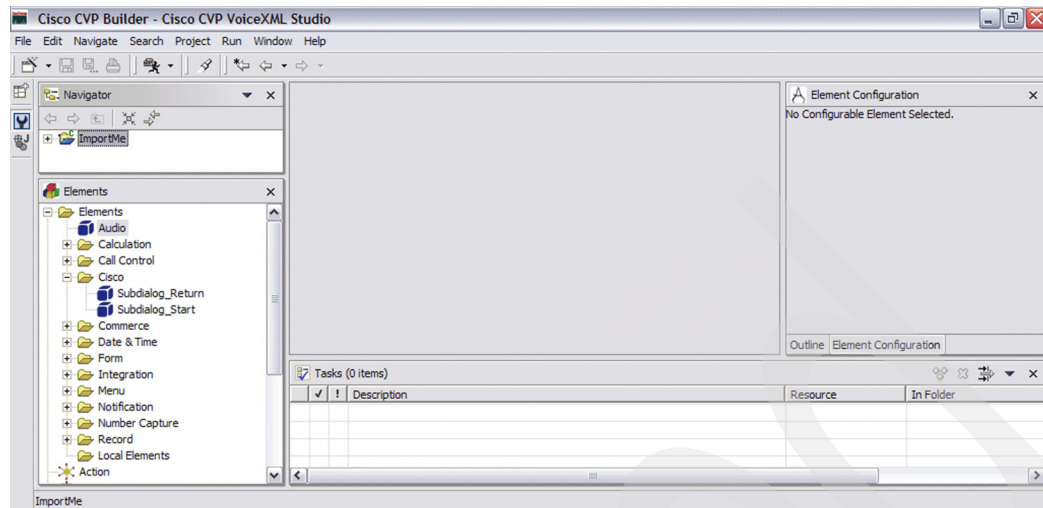


Figure 4-30 Navigator pane showing newly imported project

It is important to note that importing a project into studio only makes the contents of the project viewable from within studio. Importing does not actually copy the contents of the project directory into your default workspace directory. It is generally a good idea to keep all project directories in the workspace directory. On a default Windows installation, this directory is usually C:\Cisco\CVP\Studio\eclipse\workspace (or if you have changed the base directory from the default values INSTALLATION\_PATH\Studio\eclipse\workspace. When you import a project, copy the project directory to your workspace directory then import from that location.

### Rename an existing project

To rename a project in studio, it is not sufficient to simply rename the folder that the project resides in on the file system (because the configuration files associated with the project will not be updated appropriately to reflect the new name and you will run into problems). Follow these instructions to rename your project:

1. Right-click the project folder of the project you would like to rename in the Navigator pane and choose **Rename** as shown in Figure 4-31 on page 136.

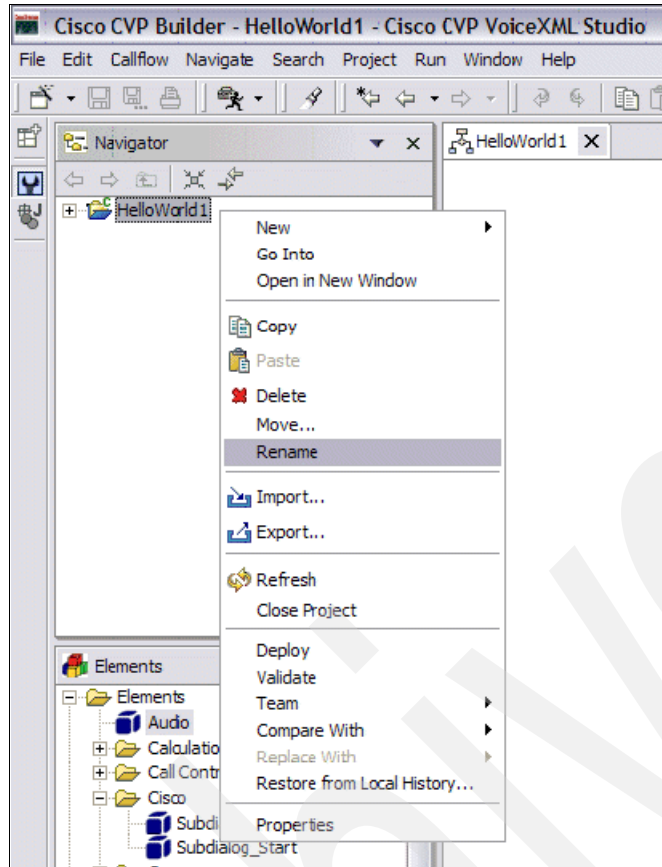


Figure 4-31 Rename a project

2. Now simply type in the new name for your project and press Enter as shown in Figure 4-32 on page 136.

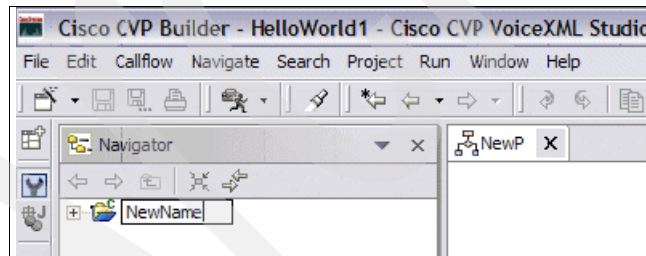


Figure 4-32 Navigator pane showing the new name of the project

## Delete an existing project

You might want to delete an existing project from your workspace view (does not delete the project folder from the file system) or entirely (does delete the project entirely) To perform either of these actions, follow these instructions:

1. Right-click the project folder for the project you would like to delete in the navigator pane and choose **Delete** as shown in Figure 4-33 on page 137.

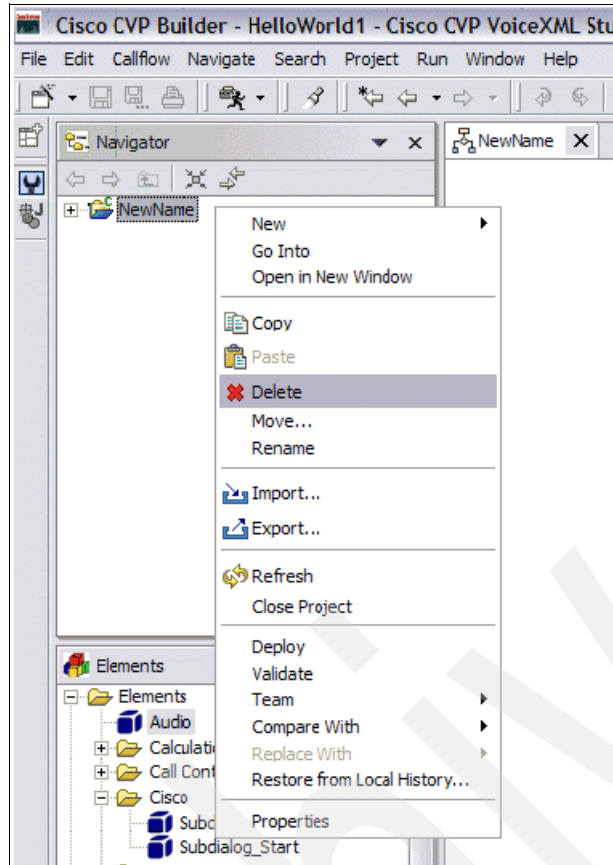


Figure 4-33 Delete a project

2. Now decide whether you want simply to delete the project from just the workspace (**Do not delete contents**) or delete the project from the workspace and the file system (**Also delete contents under C:\Cisco\CVP\Studio\eclipse\workspace\ProjectName**) as shown in Figure 4-34 on page 137.

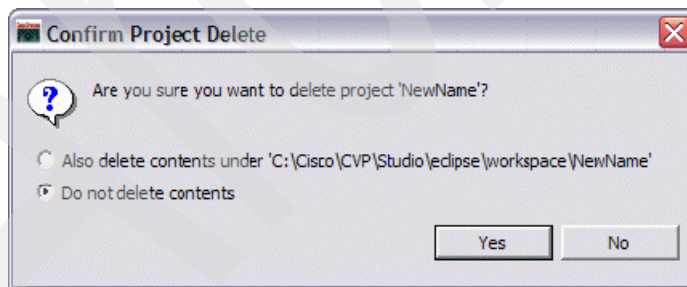


Figure 4-34 Confirm project deletion

3. Finally, click **Yes** and the project will be removed from the workspace in studio (and also from the file system if you selected **Also delete contents under**). Figure 4-35 shows the project has been removed from the workspace.

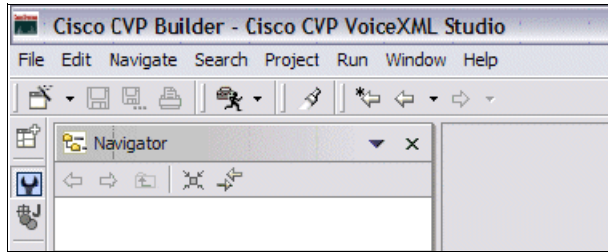


Figure 4-35 Navigator pane show that the project has been deleted

## Archiving a project

It can sometimes be very handy to have the ability to export your projects to a zipped archive (for backup purposes, for example). This can be accomplished simply by following these instructions:

1. Right-click the project folder for the project you want to export in the navigator pane and choose **Export** as shown in Figure 4-36 on page 138.

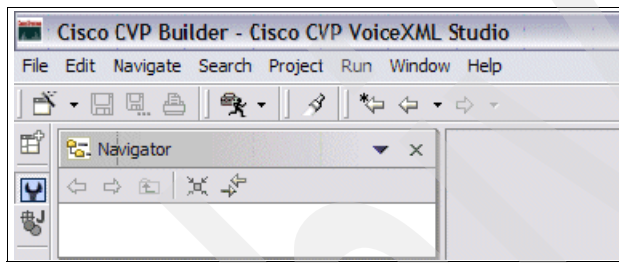


Figure 4-36 Export a project

2. Select **Zip file** and then click **Next** as shown in Figure 4-37.

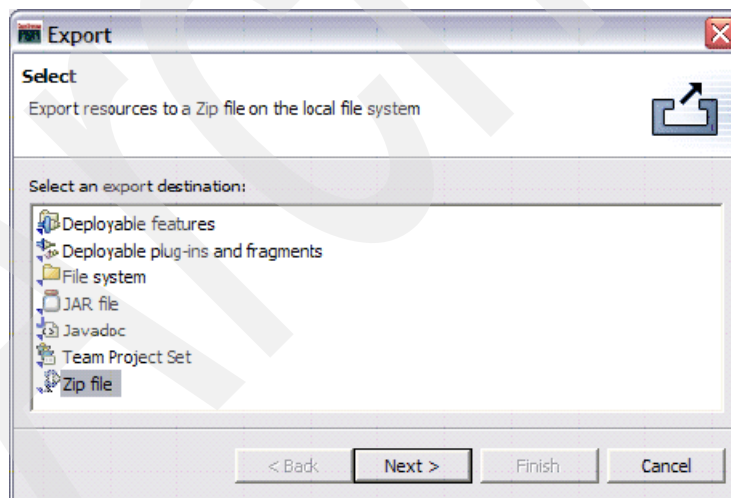


Figure 4-37 Export - Select an export destination

3. Leave all default settings and then click **Browse** to specify where you want to have the zipped archive of the project created. In this example we have chosen to save a backup of the entire project and its associated directory structure to C:\Cisco\CVP\Studio\HelloWorld1Archive.zip. Click **Finish** as shown in Figure 4-38 on page 139.

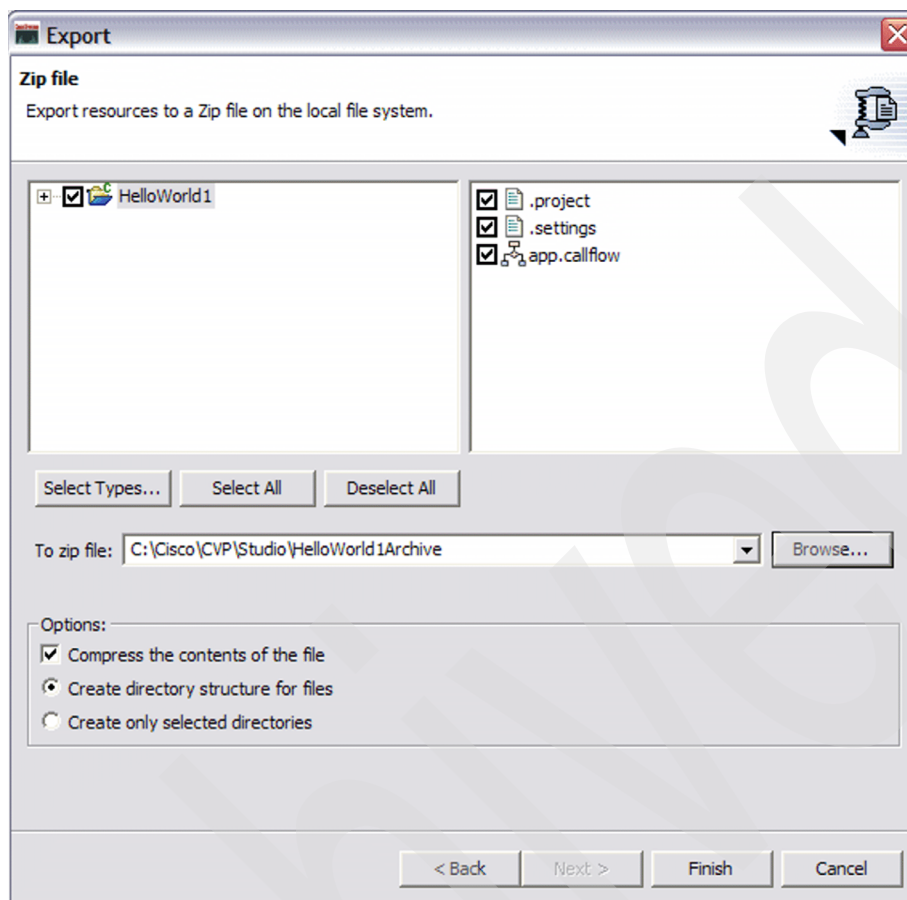


Figure 4-38 Export - specifying a ZIP file

## Modifying properties of an existing project

At times you might want to modify the project properties for an existing project (for a detailed description of various relevant project properties please see “Creating a new Cisco CVP VoiceXML Studio Project” on page 125. This can be accomplished quite easily from within studio. Simply follow these instructions:

1. Right-click the project folder of the project, the properties of which you want to modify in the navigator pane.

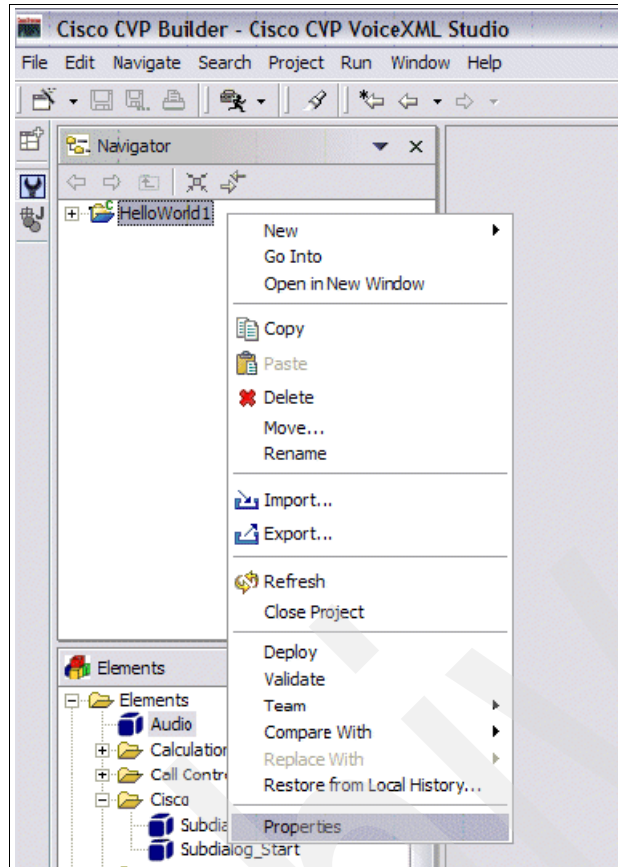


Figure 4-39 Modify the project properties

2. You can now modify project properties at will or restore default settings by clicking **Restore Defaults**. You can now see the project properties dialog box as shown in Figure 4-40 on page 140. The various property settings that can be changed are shown in Figure 4-41 on page 141, Figure 4-42 on page 141, Figure 4-43 on page 142, Figure 4-44 on page 142, and Figure 4-45 on page 142.

**Note:** The External Tools Builders and Project References tabs are quite advanced and rarely used settings and should only be modified by users who have a thorough understanding of what they are doing.

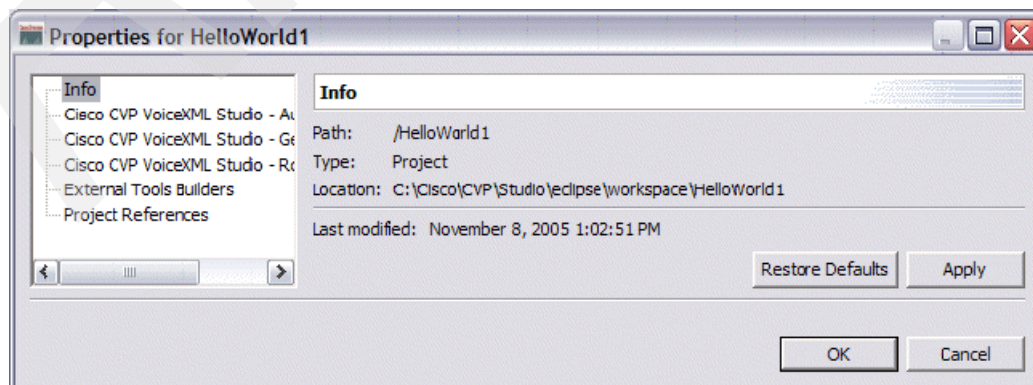


Figure 4-40 Project properties for HelloWorld1 - Info panel



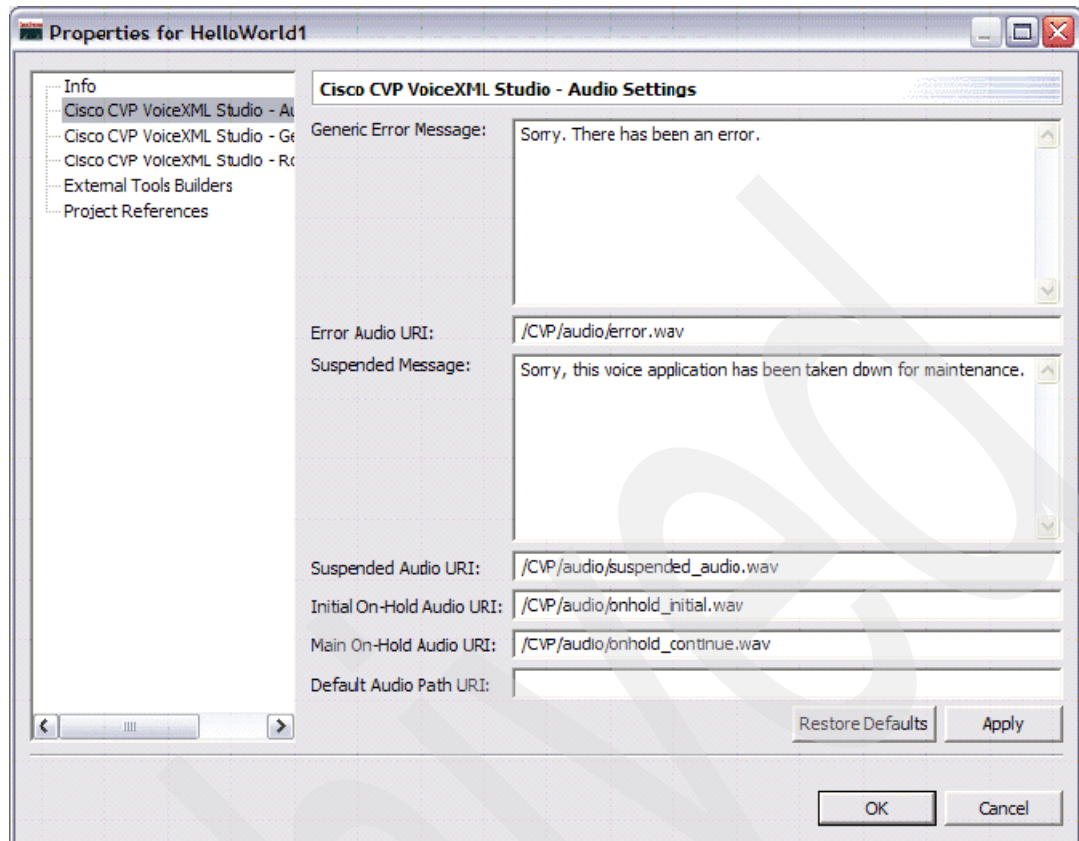


Figure 4-41 Project properties for HelloWorld1 - Cisco CVP VoiceXML Studio - Audio Settings panel

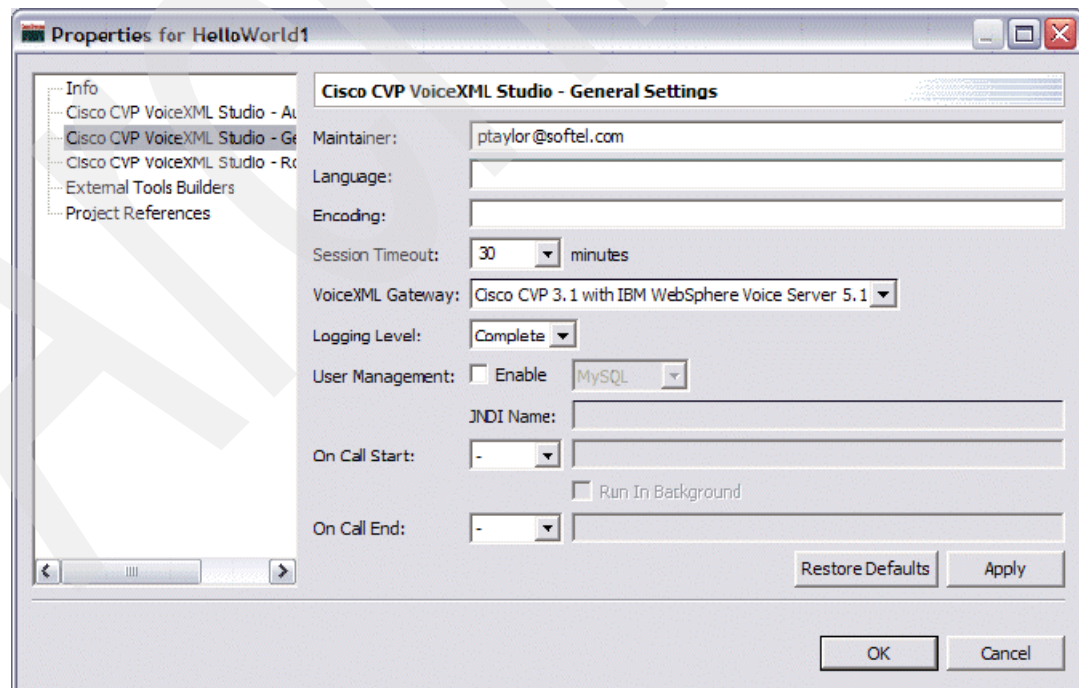


Figure 4-42 Project properties for HelloWorld1 - Cisco CVP VoiceXML Studio - General Settings panel

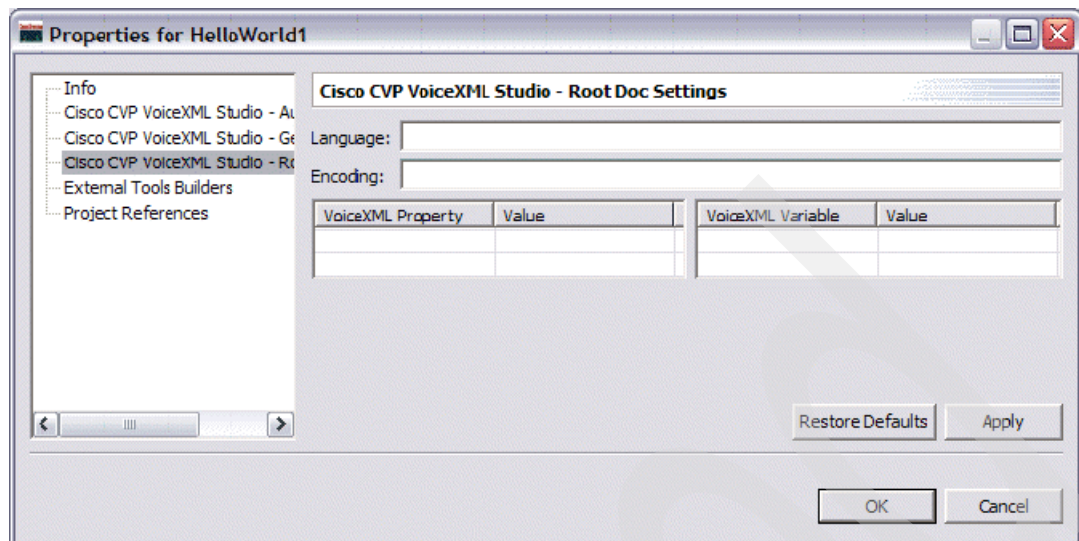


Figure 4-43 Project properties for HelloWorld1: Cisco CVP VoiceXML Studio Root Doc Settings

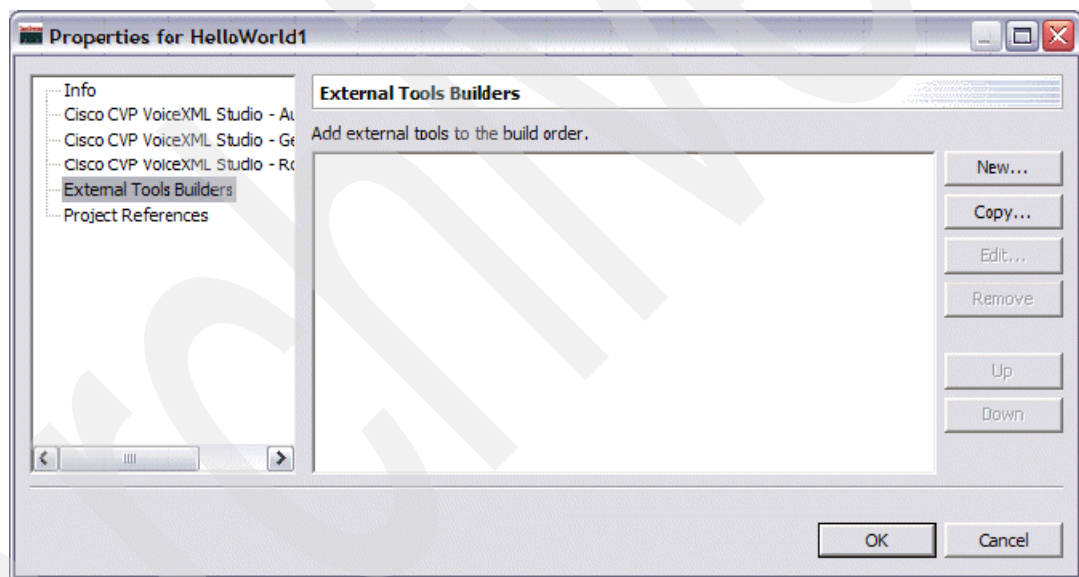


Figure 4-44 Project properties for HelloWorld1 - External Tools Builders

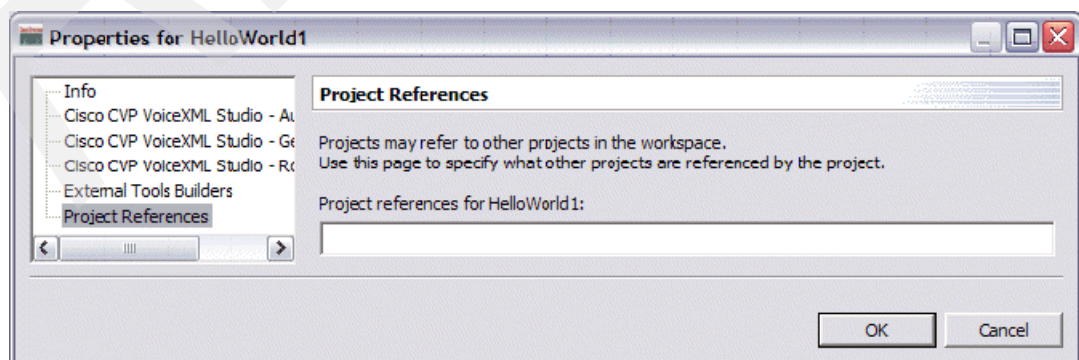


Figure 4-45 Project properties for HelloWorld1 - Project References



## Creating a trivial call flow (HelloWorld)

By now we should have a fairly thorough understanding of how to create, delete and manipulate projects in various ways that might be useful to you as a voice application developer. We can finally get into the core of application development. Let us start by taking a close look at our elegant graphical development environment so we can find our bearings.

### Cisco CVP studio workspace overview

The Cisco CVP studio workspace consists of three components (panes):

- ▶ The Element View
- ▶ The CallFlow® Editor
- ▶ The Element Configuration View

We examine these views briefly before proceeding.

#### Elements view

The Elements view displays the various components that can be employed by the developer to create voice applications (see Figure 4-46 on page 143). This pane can be thought of as a toolbox with elements that can be dragged into the call flow editor pane for the purpose of creating detailed call flows with a drag-and-drop paradigm. This pane contains a wide variety of configurable elements designed by Cisco and other third-party developers. You can also have your own custom configurable elements appear in this pane under the Local Elements folder. For a brief description of the elements seen in this pane, see Section 4.3, “Elements in studio” on page 105.

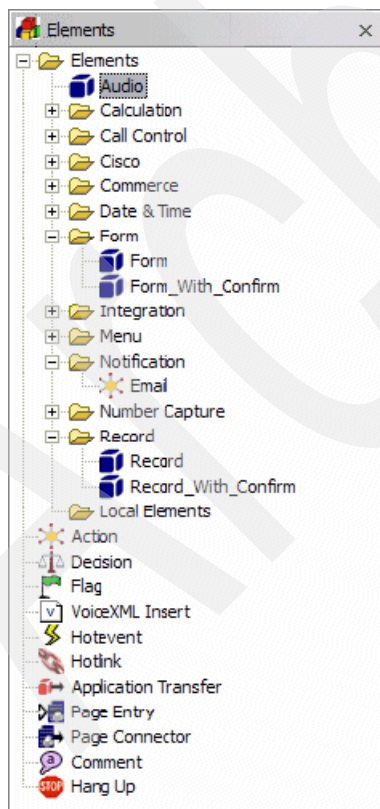


Figure 4-46 Elements view

### ***Element configuration view***

The Element Configuration view is the pane in which modifications can be made to an element that is selected in the Call Flow Editor pane. This panel appears differently, according to what type of configurable component is currently selected.

Figure 4-47 shows the element configuration view for an Email element.

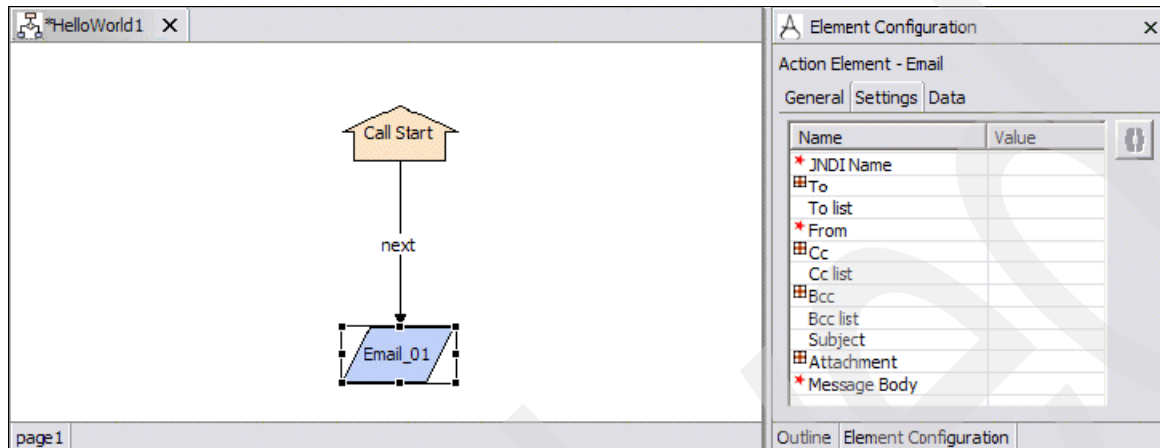


Figure 4-47 Element Configuration pane - Action Element - Email

### ***Call Flow Editor***

The Call Flow editor is where we build the application with a drag-and-drop paradigm. This is accomplished by dragging the desired elements onto the pane and then connecting their exit states in accordance with the business logic we want to implement in our application. The end result of our drag-drop-and-connect process is an application call flow that resembles a flow chart that lays out the logic that will be executed at runtime. Before we go into the creation of our HelloWorld application let us examine a few key concepts that are employed continually while working in the call flow editor.

### **Creating the HelloWorld application call flow**

The next few sections demonstrate how to create an application call flow.

#### ***Resizing and positioning elements***

After an element has been dragged into the editor pane, it can be moved and resized by selecting the item and then dragging appropriately. Also note that multiple items can be selected simultaneously by holding down the CTRL key and clicking on the items you want to select. A similar effect can be achieved by holding down the left mouse button and then dragging across all desired elements to create a selection rectangle. All selected elements can then be repositioned simultaneously. The next seven diagrams (Figure 4-48 on page 145 to Figure 4-54 on page 148) illustrate this concept.

1. To resize an element, you must first select it as shown in Figure 4-48 on page 145.

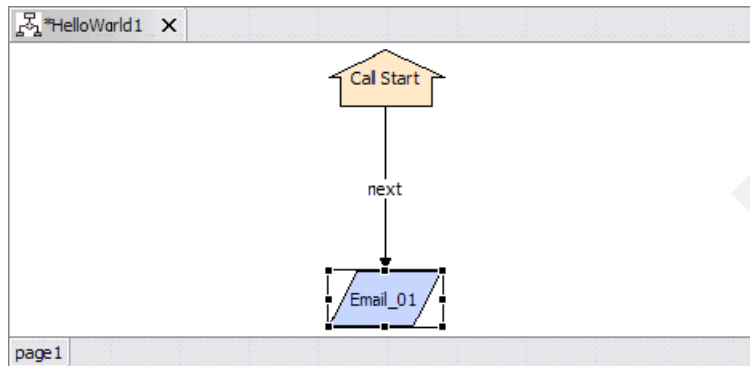


Figure 4-48 Resizing an element - start by selecting it

2. Drag on the corner of the element until you achieve the desired height and width (see Figure 4-49 on page 145).

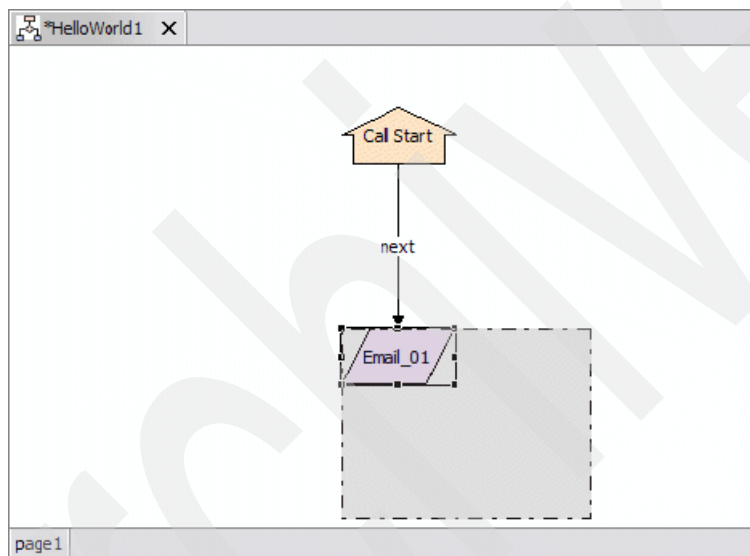


Figure 4-49 Resizing an element - dragging the corner

3. The element is now resized as shown in Figure 4-50 on page 146.

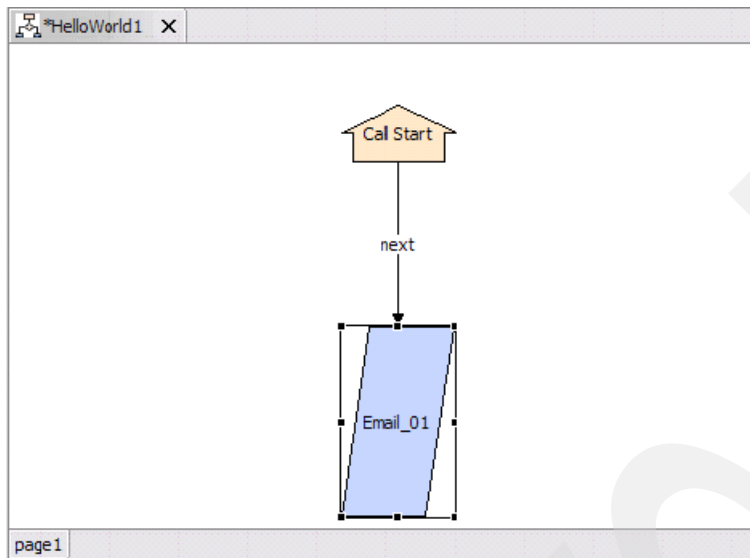


Figure 4-50 Resizing an element - the newly resized element

4. To move an element, simply select it in the editor pane (Figure 4-48 on page 145) and then drag it to where you would like it to be as shown Figure 4-51.

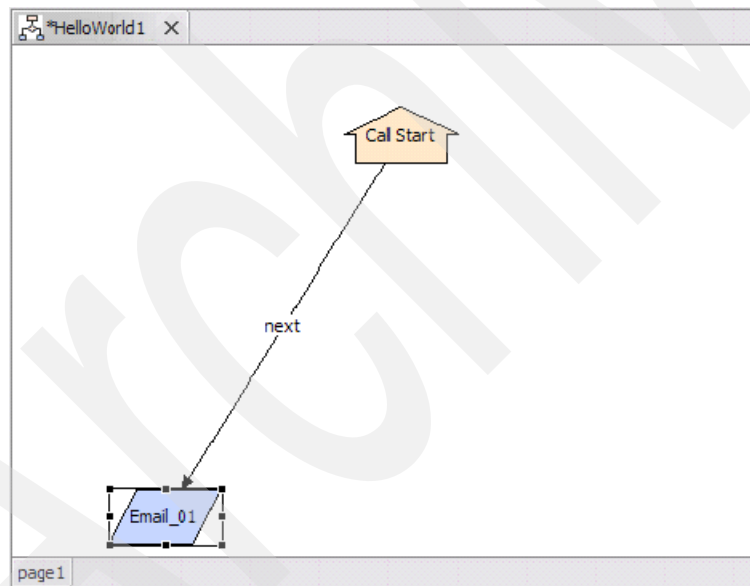


Figure 4-51 Moving an element - dragging it to its new location

5. Select multiple items by dragging a marquee around the desired items as shown in Figure 4-52.

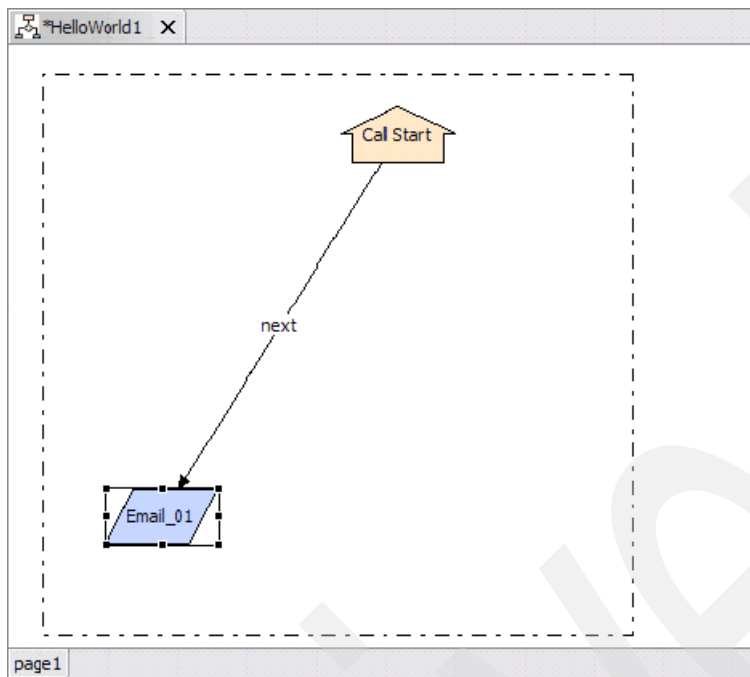


Figure 4-52 Selecting multiple items by dragging a marquee around them

6. Multiple items can then be repositioned simultaneously as shown in Figure 4-53 on page 147.

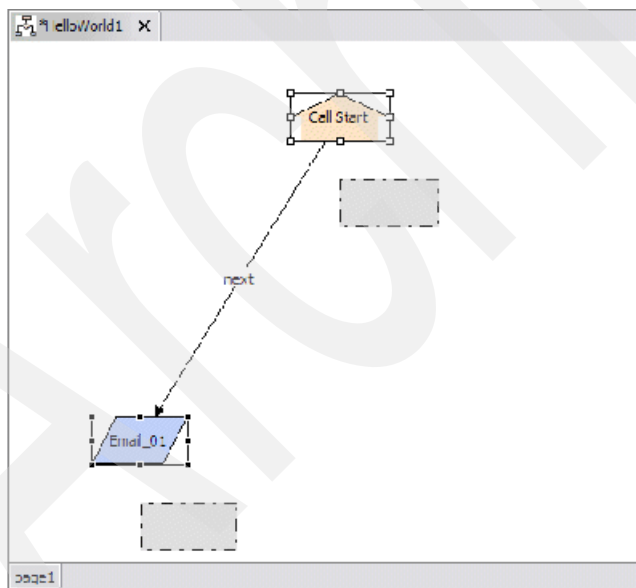


Figure 4-53 Repositioning multiple items

### **Naming elements**

Elements can be named and renamed as many times as the developer wants. An element dragged onto the editor pane will be given a default name by studio (for example, Form\_09). This can be modified by the developer as he or she sees fit, as long as there are no duplicate element names within the context of the application.

1. Right-click the element and select **Rename** as shown in Figure 4-54 on page 148.

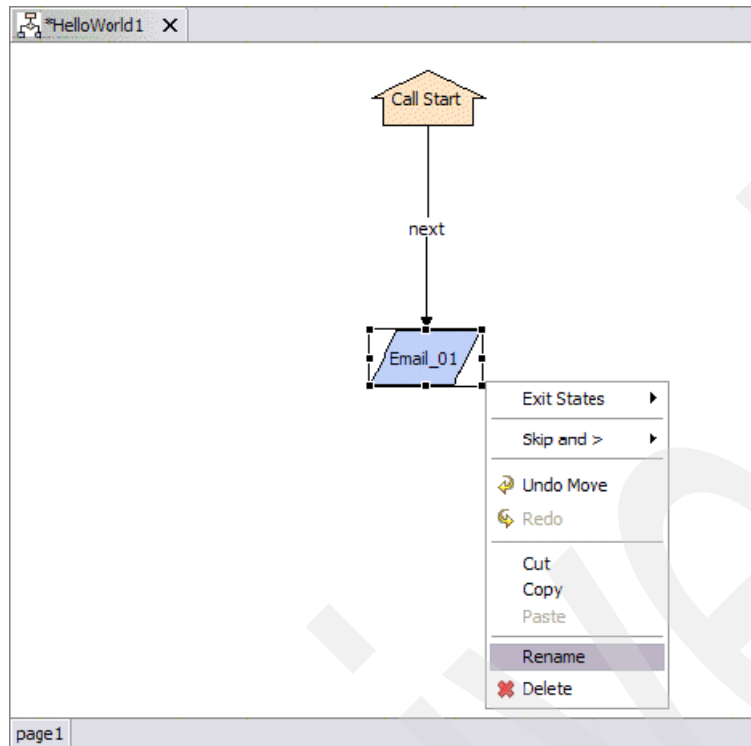


Figure 4-54 Renaming an element using the context menu

2. Now type in the new name for the element and then press Enter as shown in Figure 4-55 on page 148.

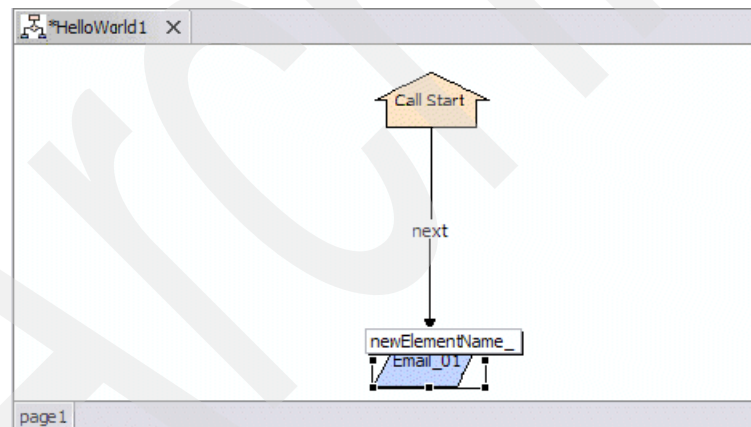


Figure 4-55 Typing the new element name

### Some notes on adopting a consistent element nomenclature

As the complexity of your projects grow and as you are forced to reference element names programmatically in order to access element and session data, you will find that adopting a consistent element nomenclature can be extremely helpful in avoiding confusing applications. We have adopted the following naming convention while developing applications for this Redpaper:

*Approximate description of function + “\_” + element type*

That is to say, we name our elements by taking a succinct description of their function in the call flow, an underscore character, and then the element type. So for example, if we have a Form\_With\_Confirm whose purpose it is to get the name of a color from the user, we would name it something like:

```
getColorName_formWithConfirm
```

We cannot emphasize enough how this makes application call flows more developer friendly (or at least human readable). Naturally, if the names get ridiculously long or if this nomenclature does not fit with your personal style, feel free to adopt your own. Just ensure that it is consistent and understandable so that when you or your team mates go back to modify your application weeks, months, or years from now they will be intelligible.

### **Connecting elements**

Elements in the call flow are connected by right-clicking the element, selecting the exit state from the context menu list, and then dragging the line to the element to which you want to connect it.

1. Right-click the element and select the exit state that you would like to link with another element as shown in Figure 4-56.

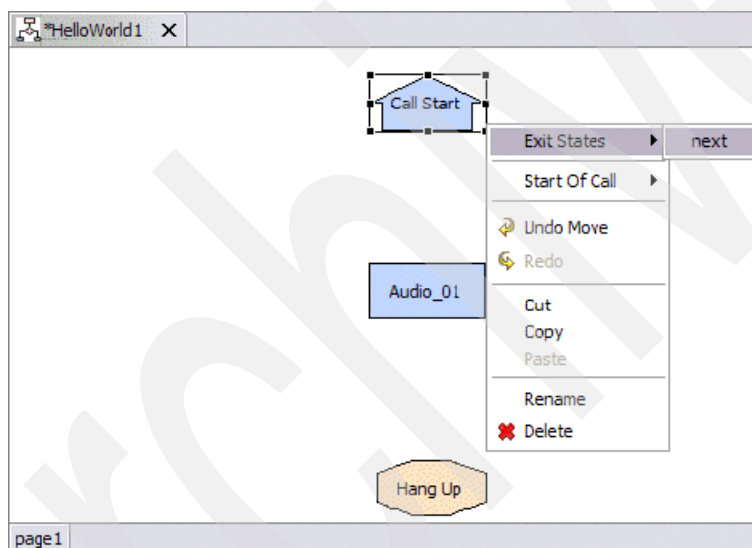


Figure 4-56 Connecting elements using the context menu

2. Now drag the exit state to the element you would like connect as shown in Figure 4-57.

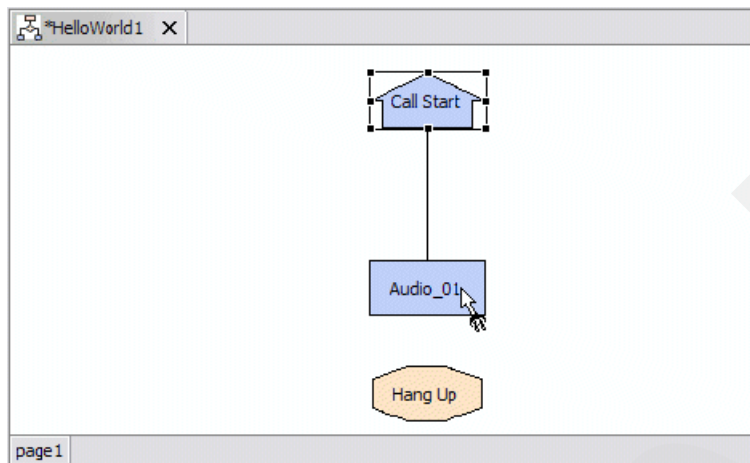


Figure 4-57 Connecting the exit state to another element

The two elements are now connected as shown in Figure 4-58 on page 150.

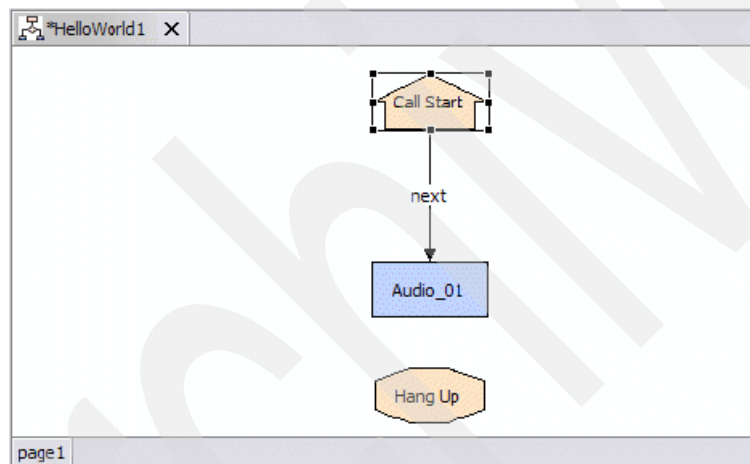


Figure 4-58 Connected elements

### ***Adding and deleting angles and bends***

New bend points can be added to the connections between elements by simply positioning the cursor over the midpoint of the connection, selecting the connection and then dragging. This process can be repeated as many times as is desired to produce the appropriate number of angles and bends. An angled line can be deleted by selecting it and then pressing **Delete**.

Lines connecting elements can be bent to increase readability of call flow diagrams, as shown in Figure 4-59.



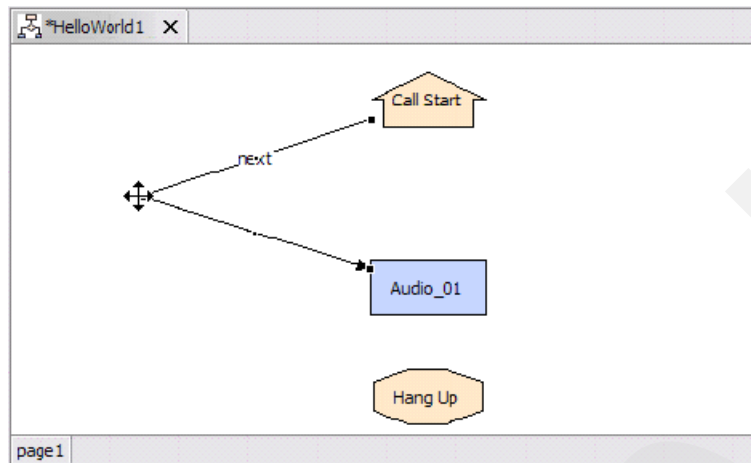


Figure 4-59 Bending a connecting line for readability

### ***Skipping elements***

When designing a voice application, it is often desirable to begin testing a portion of an application while skipping over portions that are not yet complete or fully functional. This incredibly useful feature is achieved easily in the call flow editor. All that needs be done is to right-click the element that you want to skip in the editor pane and select **Skip and >**. There are two possible ways to skip an element. The first is to **Ignore (Orphaned)** where the element is treated as an orphan and ignored altogether. The other is to **Skip and > → Exit as ...** where the ellipses indicate any one of the possible exit states associated with the element in question.

In this scenario (see Figure 4-60 on page 152), our application starts off with a decision that can lead to two possible branches in the call flow. Let us assume that the first branch (the one lead to by the `execution_branch_1` exit state for `Decision_01`) is fully functional and we would like to test it and also that the branch lead to by `execution_branch_2` has a glitch in it that raises an unhandled exception.

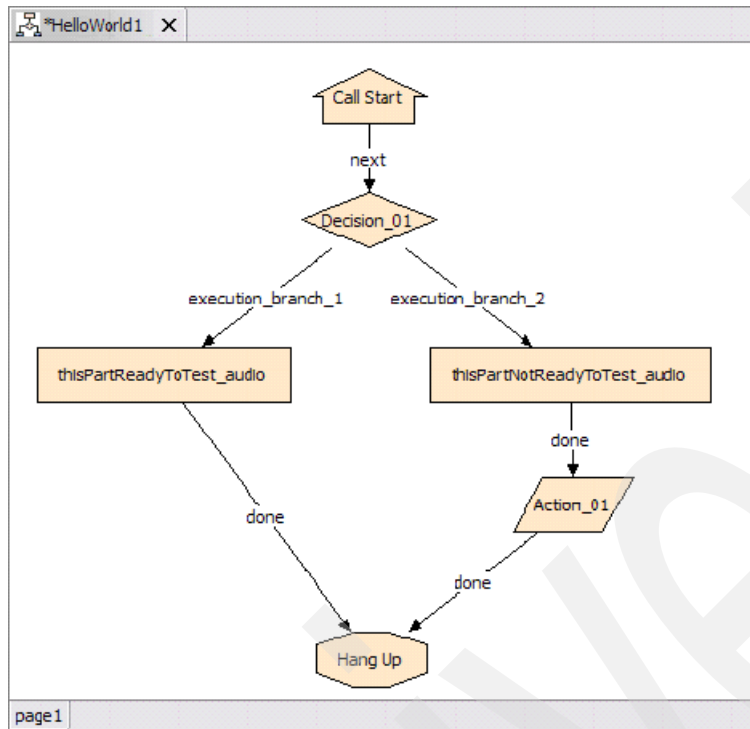


Figure 4-60 Example of skipping elements

Now, we decide to skip Decision\_01 and always exit with the exit state execution\_branch\_1 as shown in Figure 4-61 on page 152.

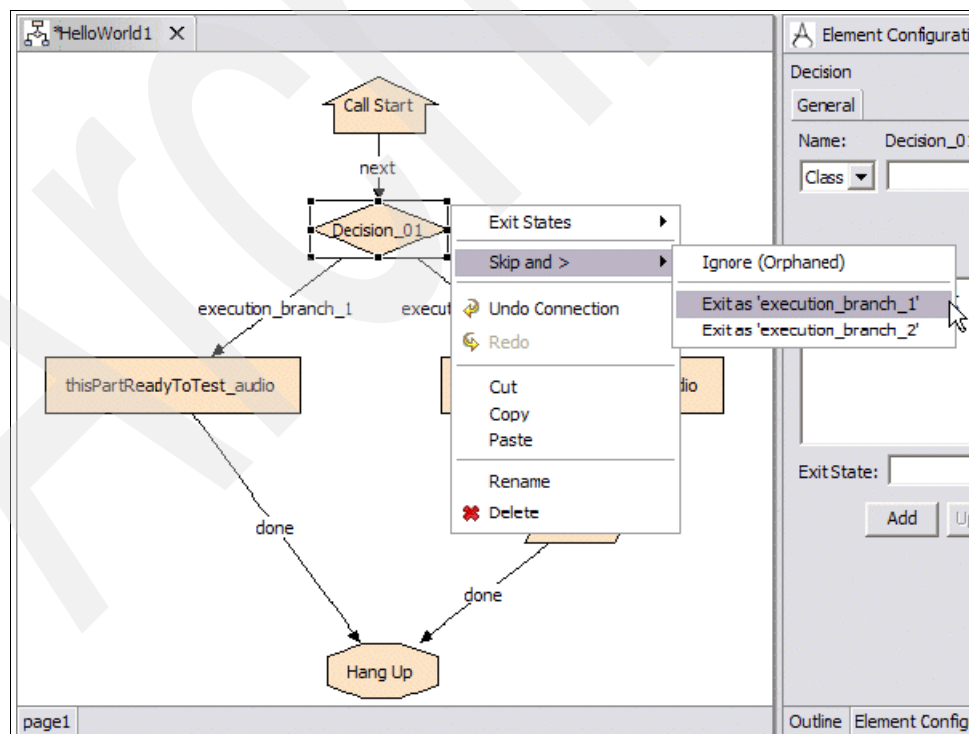


Figure 4-61 Skipping Decision\_01 and exiting with the exit state of execution\_branch\_1

The logic defined in the second execution branch will never be executed (see Figure 4-62 on page 153), so we know that we can safely deploy our application and see what happens when the decision element exits with `execution_branch_1`.

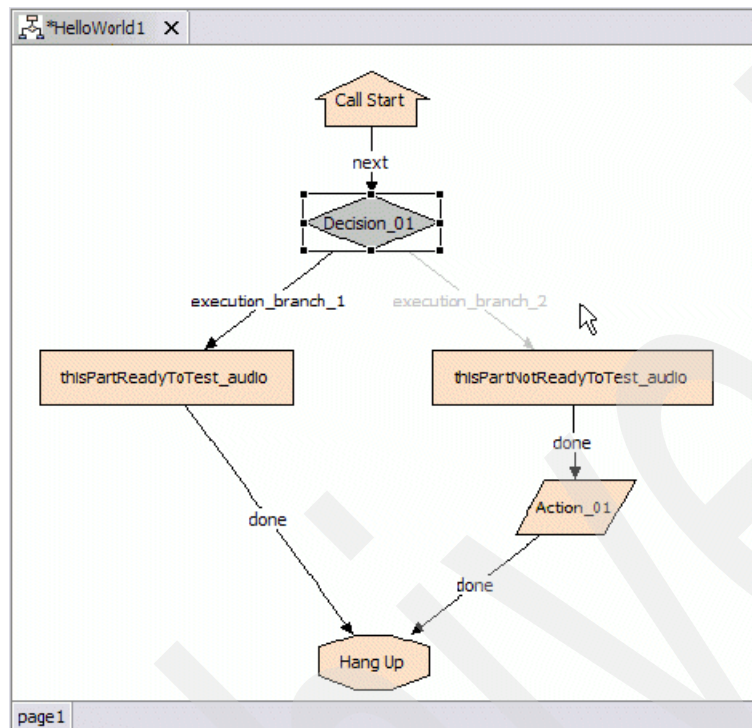


Figure 4-62 *execution\_branch\_2 will never be executed*

Now if we no longer wish to skip and always exit as `execution_branch_1`, we right-click the element and choose **Skip and > → Don't Skip**. Our call flow reverts back to its original state as shown in Figure 4-63 on page 154.

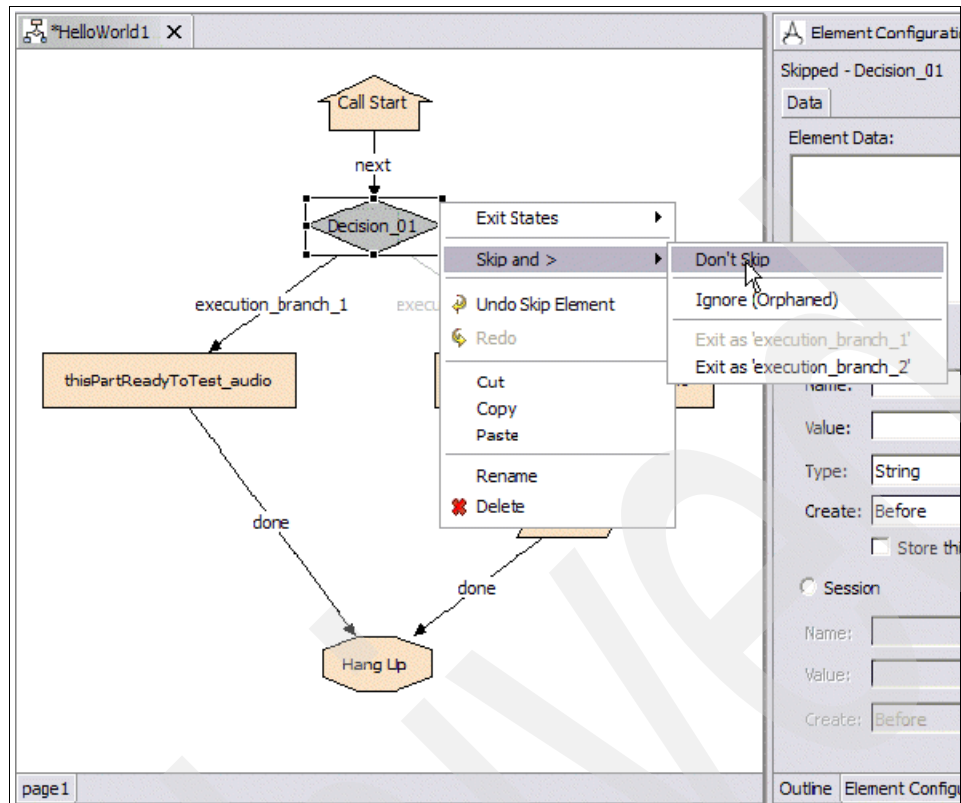


Figure 4-63 Reverting back to the original state

Now, let us assume that most of execution branch 2 is functioning with the exception of Action\_01, which we want to ignore as shown in Figure 4-64 on page 154.

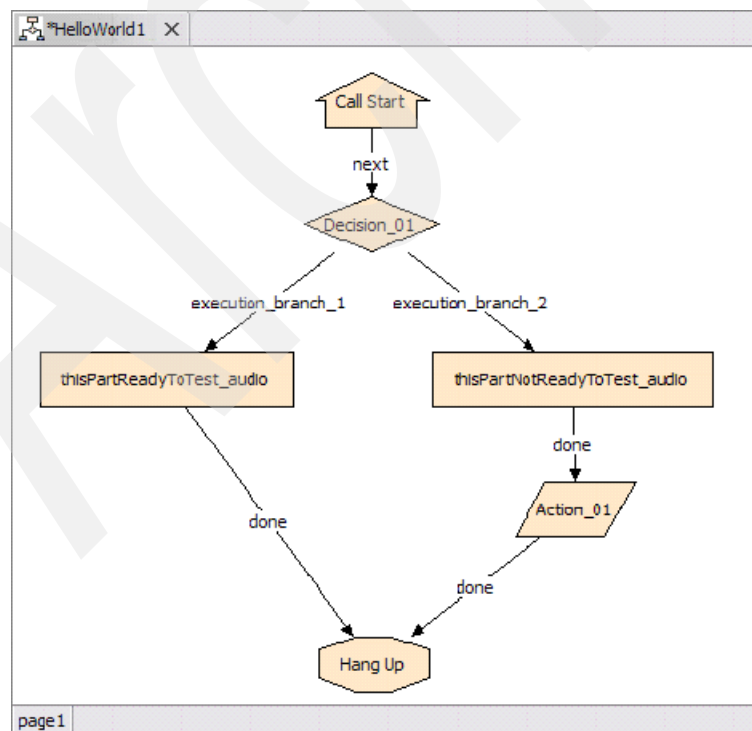


Figure 4-64 execution\_branch\_2 with Action\_01

We can remove Action\_01 from the equation entirely by orphaning the element and bypassing it as shown in Figure 4-65 on page 155.

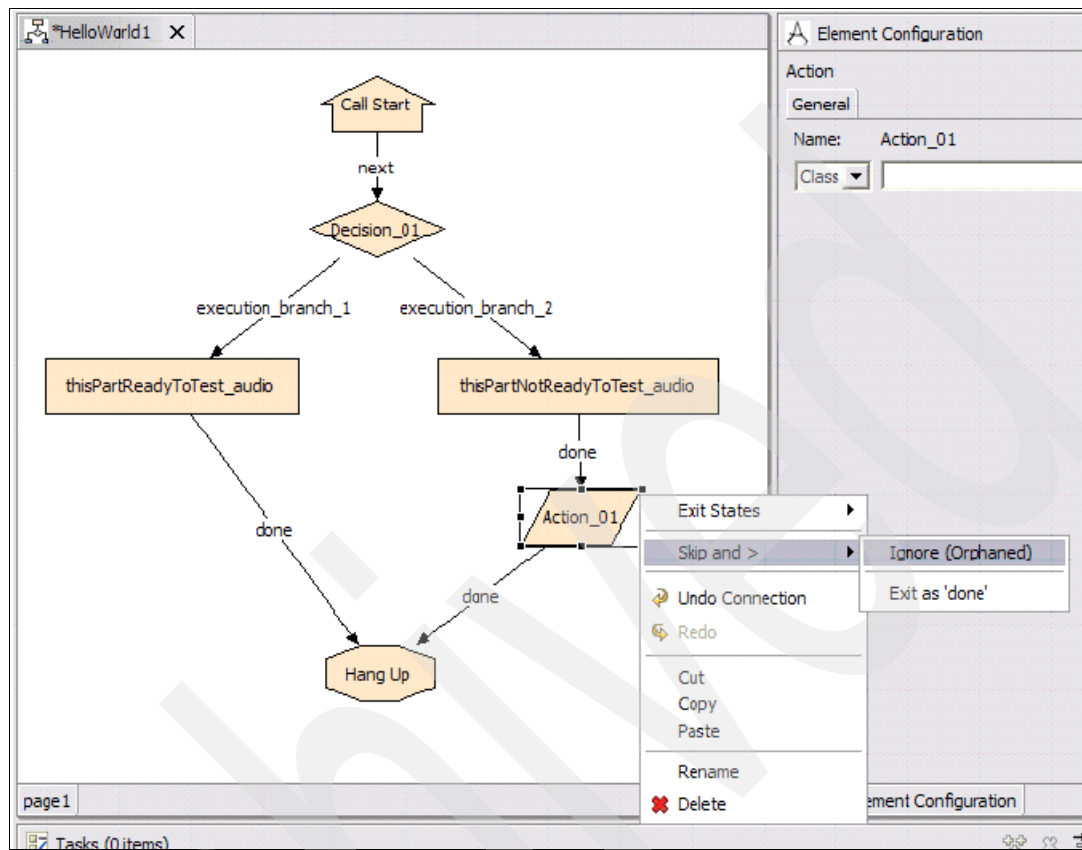


Figure 4-65 Ignoring an action using the context menu

Now, Action\_01 is no longer active in the call flow (see Figure 4-66 on page 156). But our application will no longer work (more on this later). We now have to connect thisPartNotReadyToTest\_audio's exit state to the hang up so our application will validate and be runnable.

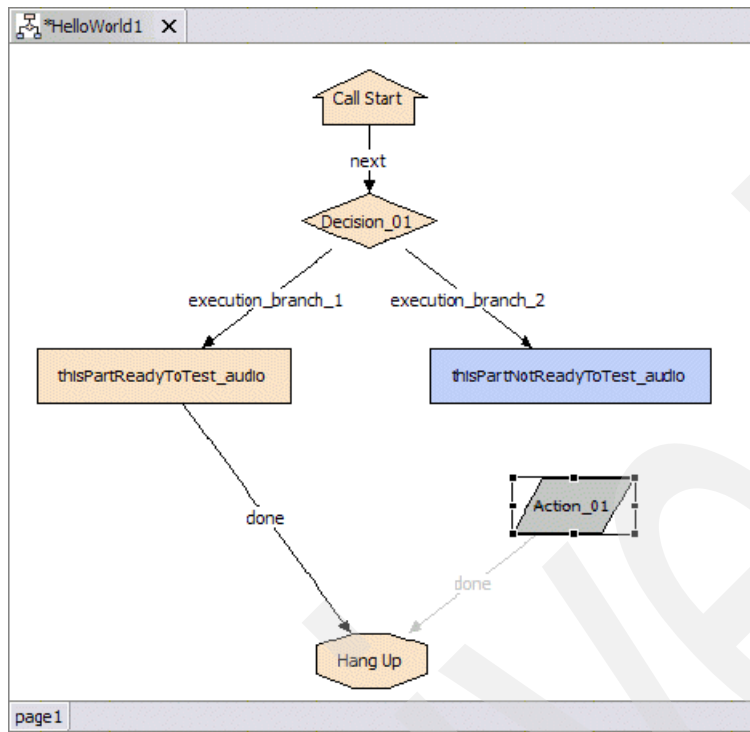


Figure 4-66 Action\_01 is no longer active in the call flow

We have now completely bypassed Action\_01 and have a runnable application (see Figure 4-67 on page 157). When we want to bring Action\_01 back into play (e.g. if we resolve whatever problems we were experiencing) we can simply reconfigure the call flow to its original state and redeploy.

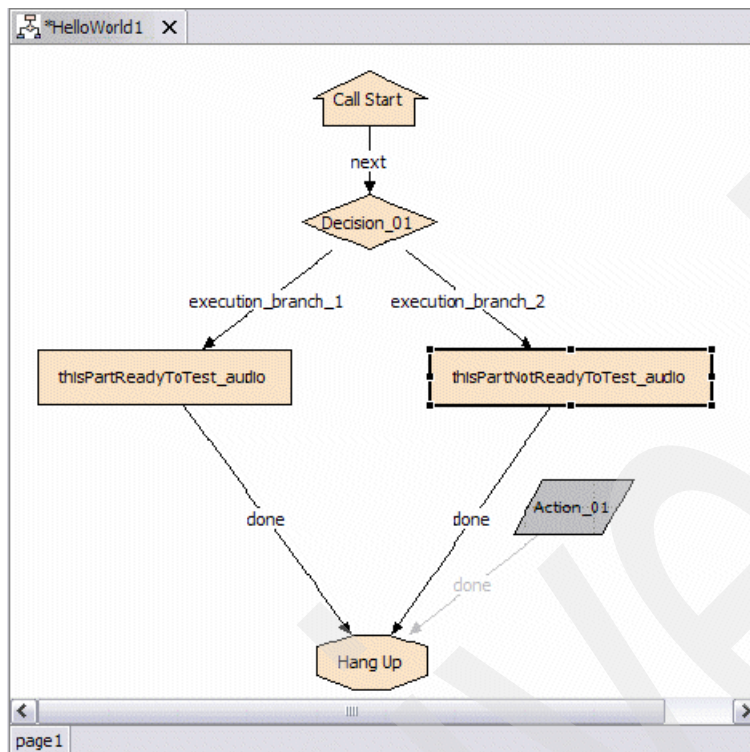


Figure 4-67 Action\_01 is ignored

### ***Copying, pasting, deleting elements from call flow + undo and redo***

Copying, pasting, deleting, undoing, and redoing all work as you would expect them to work. Elements can be copied, cut, pasted, and deleted along with all of their connections by right-clicking the element and selecting the appropriate action or doing so from the Edit menu.

Figure 4-68 on page 158 shows an example of copying an element using the context menu.

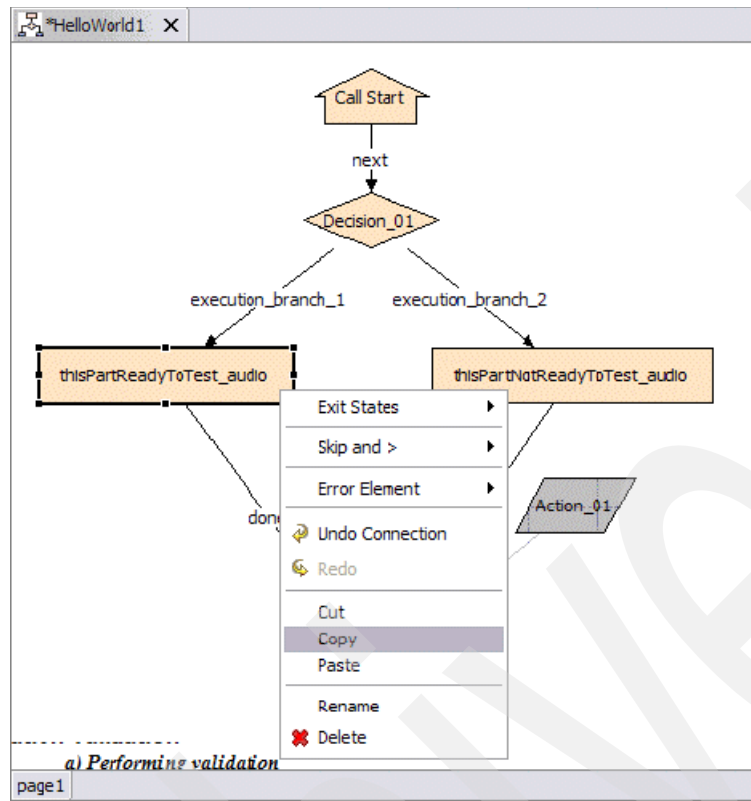


Figure 4-68 Copying elements using the context menu

Figure 4-69 on page 159 shows an example of pasting an element into the call flow.



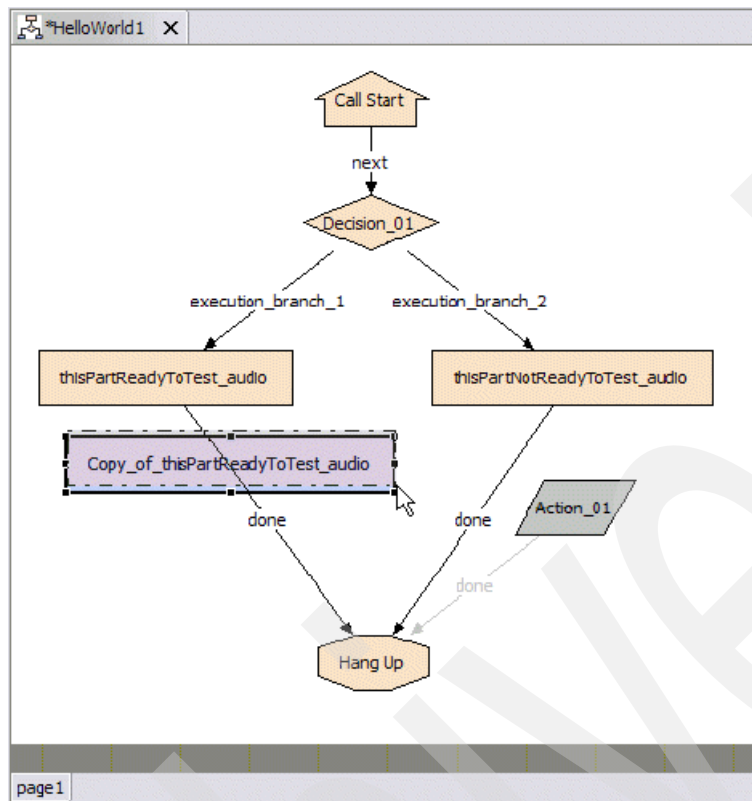


Figure 4-69 Pasting an element

Figure 4-70 on page 160 shows an example of cutting an element from the call flow using the Edit menu.

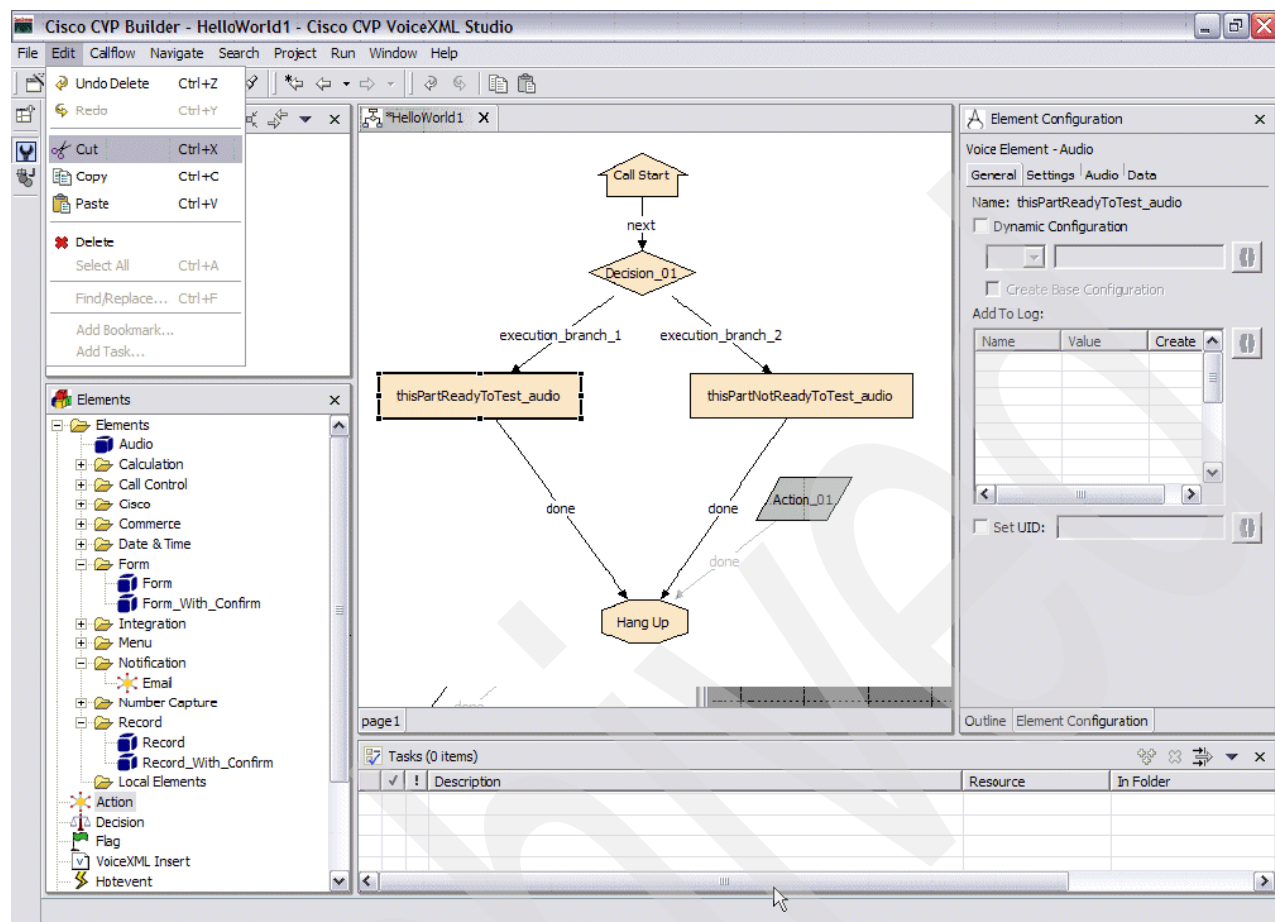


Figure 4-70 Cutting an element using the Edit menu

### Zooming in and out

You can zoom in and out within the call flow editor view.

Right-click empty space in the call flow editor and select either **Zoom In** or **Zoom Out** as shown in Figure 4-71 on page 161.

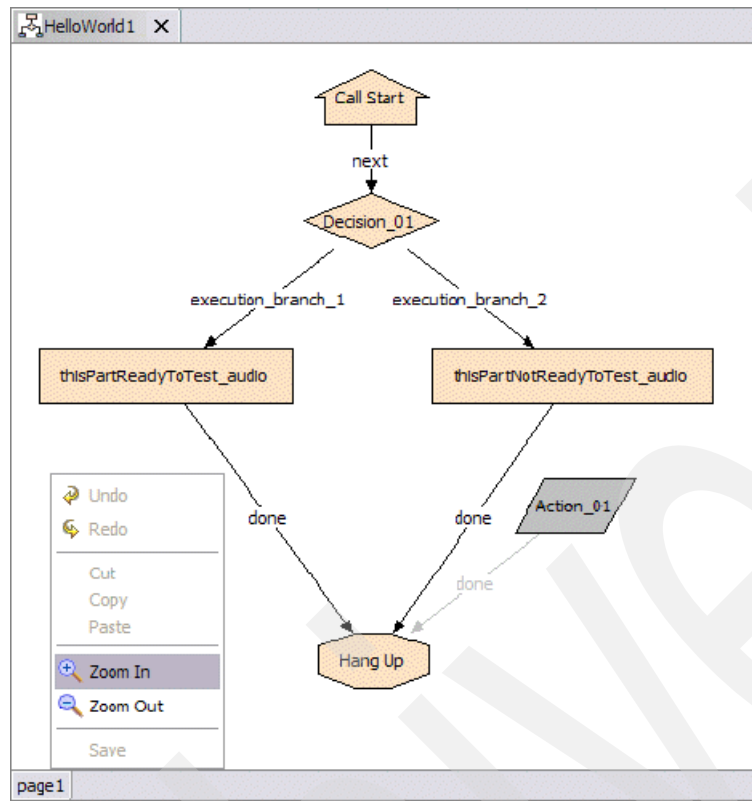


Figure 4-71 Zooming In on an element

Figure 4-72 on page 162 shows the zoomed in view of the call flow elements.

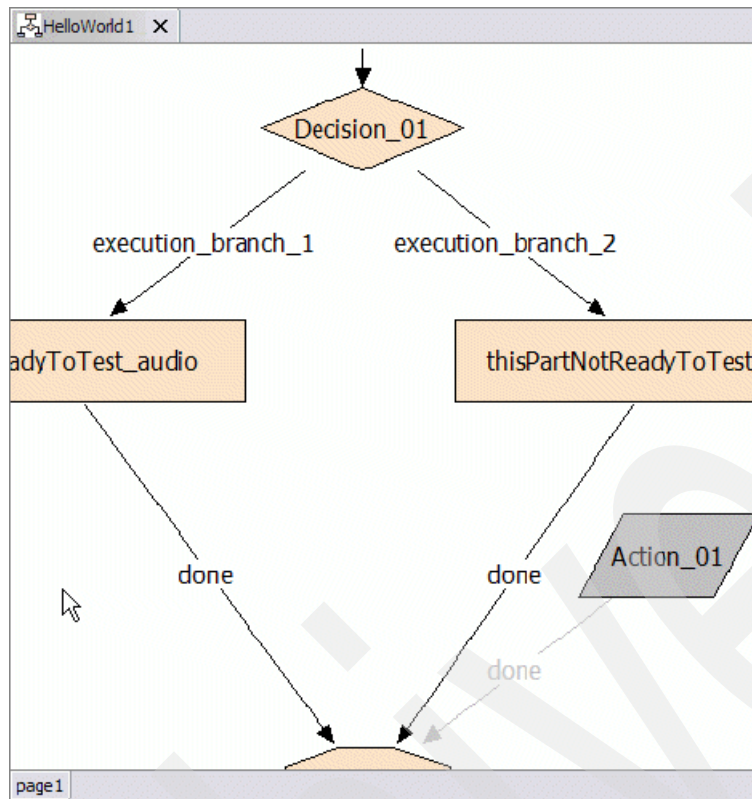


Figure 4-72 Zoomed In view

Figure 4-73 on page 162 shows a zoomed out view of the call flow.

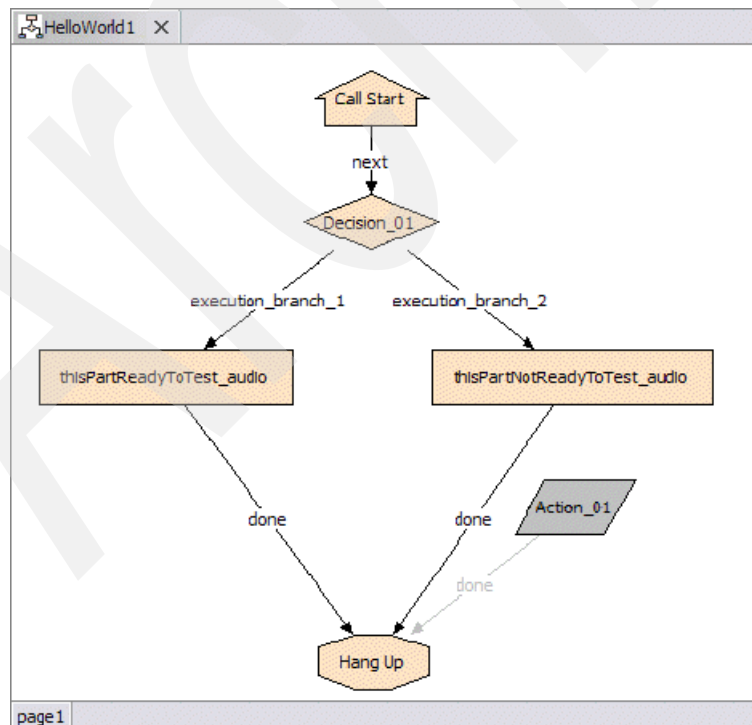


Figure 4-73 Zoomed Out view

We have now covered all the concepts you require to navigate ably through the call flow editor interface. Now let us discuss our first and simplest application. The application will be called HelloWorld1. We have already created our HelloWorld1 project and set all the appropriate properties for our project. Now we open up app.callflow and create a call flow for our application as shown in Figure 4-74 on page 163.

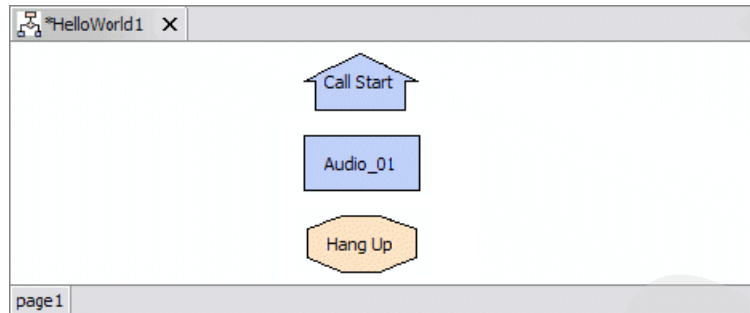


Figure 4-74 Application call flow

So in the editor pane we have the following:

- ▶ Call Start element (this should be there by default)
- ▶ A single Audio element (this will be used to read our TTS string to the user)
- ▶ A single Hang Up element

1. We will do a bit of configuration on the Audio element. First , give it an intelligible name.

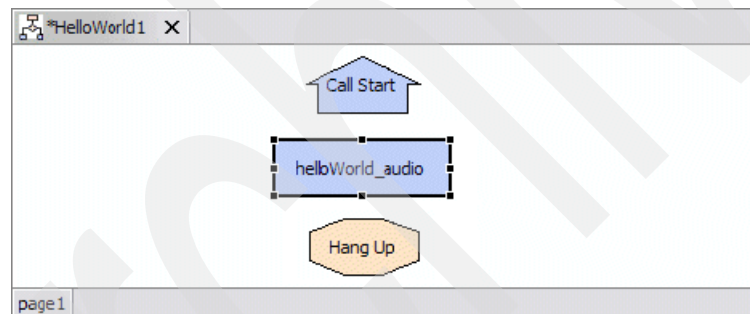


Figure 4-75 Configuring the audio element

2. Configure the audio properties for the helloWorld\_audio audio element so that it reads the TTS string Hello world to callers who access the application as shown in Figure 4-76.

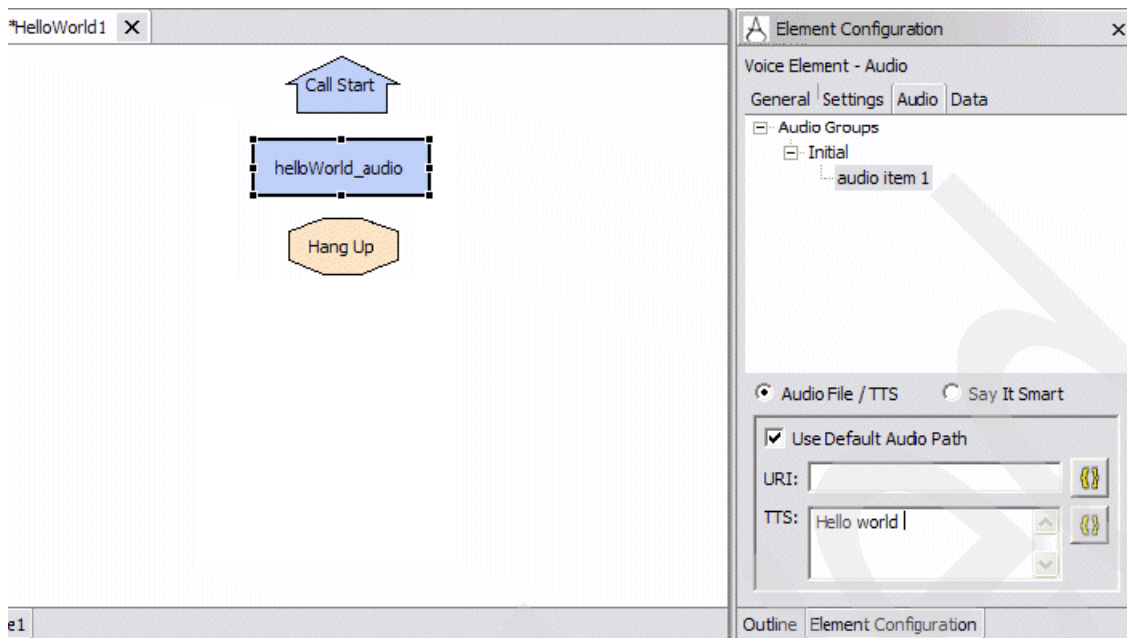


Figure 4-76 Element configuration - Voice Element - Audio

We have now defined the TTS content for the initial audio group (the first audio group played when a caller comes across the element in the call flow). That is all we need to do to configure a simple audio element.

3. Now, connect the exit states for all the elements so you have a coherent sequencing of events in the call flow (Call Start → helloWorld\_audio → Hang Up) as shown in Figure 4-77 on page 164 and Figure 4-78 on page 165.

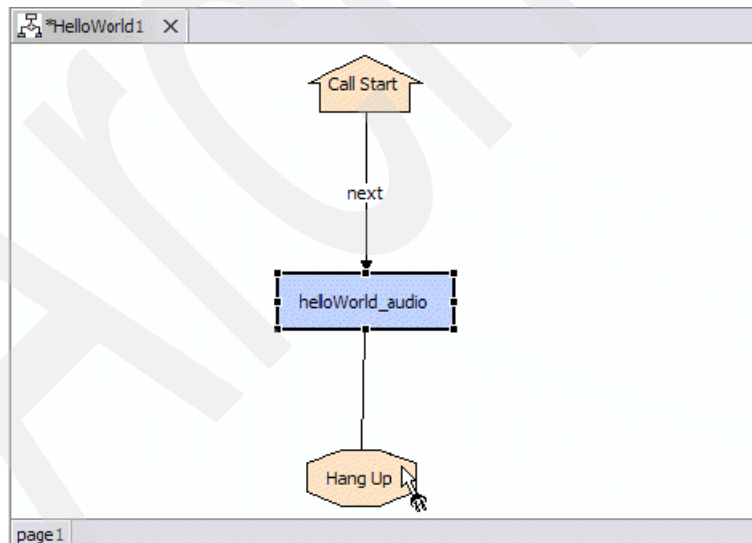


Figure 4-77 Connecting the exit states

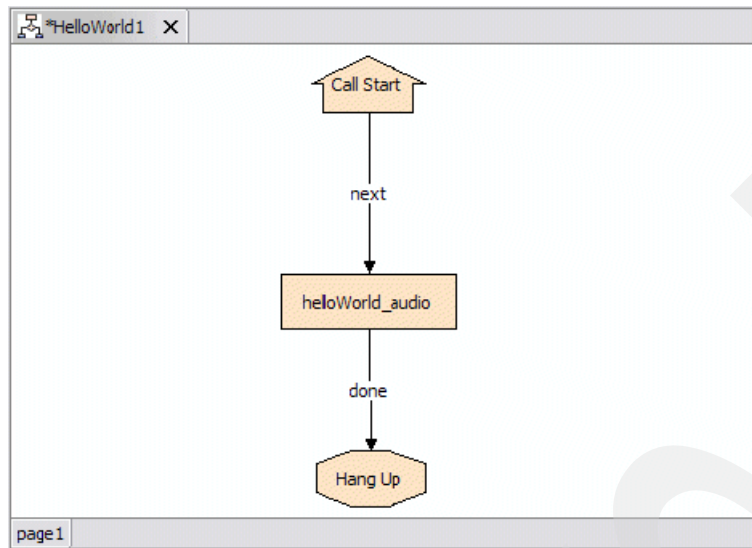


Figure 4-78 The exit states are connected

Notice that we now have a call flow where all elements are a pleasant beige shade. This is an indication that we have connected correctly all exit states for all elements in the call flow and we are almost ready to validate and deploy our application for testing.

Before you go any further, save your work. You know your current changes are unsaved when you see an asterisk next to the project name in the call flow editor as shown in Figure 4-79. Also notice that the save button is not grayed out when there are unsaved changes. Save yourself a lot of heartache and save your changes regularly as you progress in your development work.



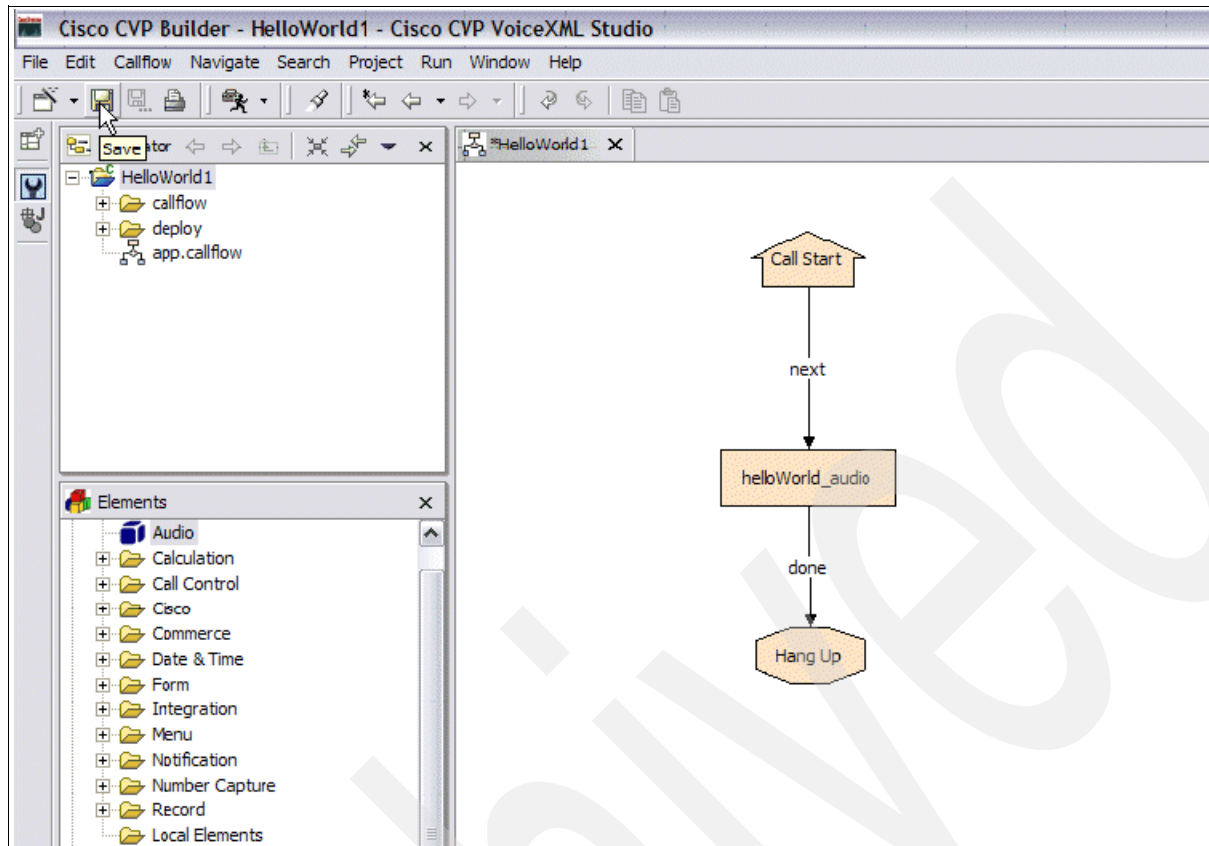


Figure 4-79 Adding a comment to the code

Figure 4-80 on page 166 shows a comment being created to document the call flow.

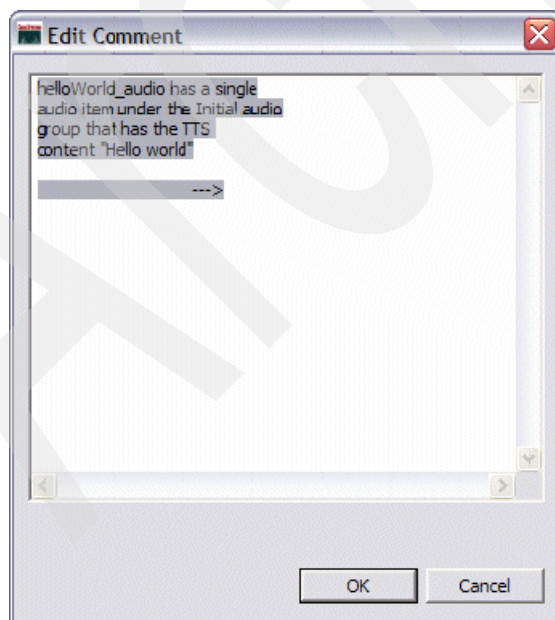


Figure 4-80 Editing a comment

Figure 4-81 shows the comment in the call flow.



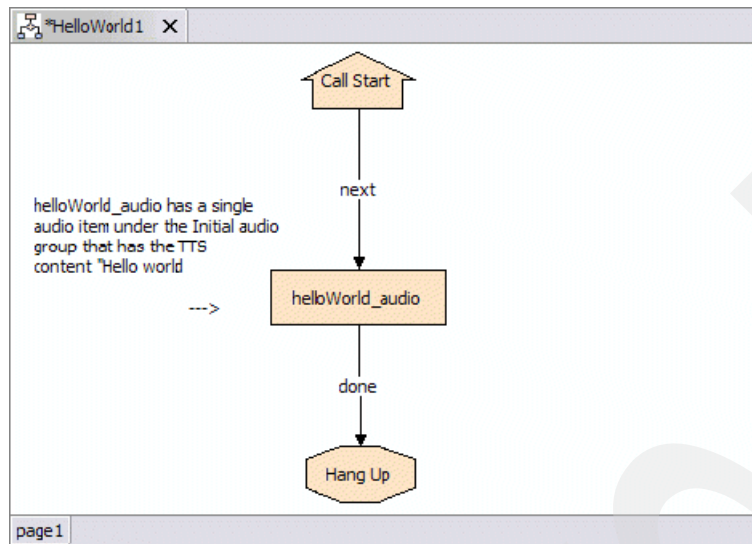


Figure 4-81 Call flow with a comment

### Application validation

We have now arrived at the point where we have fully elucidated the call flow for our HelloWorld1 application. We are now ready to validate our application in order to assure that there are no errors present that will prevent us from deploying the application successfully. Validation is the equivalent of checking for compilation errors. Validation enables you to catch errors that prevent the successful compilation and deployment of your application to the Cisco CVP application server but does not catch any logical or run-time errors. First, we examine what happens upon a successful validation.

In order to validate the application, right-click the project folder for the project you want to validate in the Navigator pane and select **Validate** as shown in Figure 4-82 on page 168.

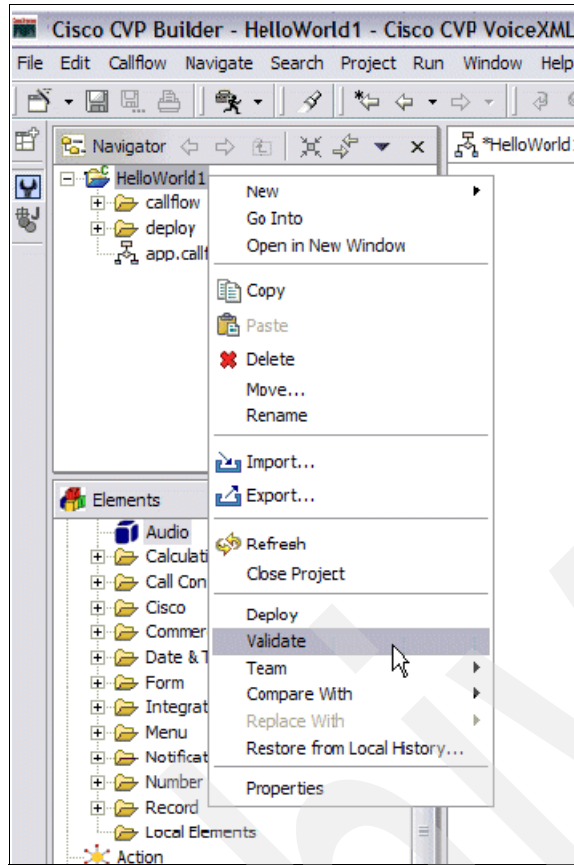


Figure 4-82 Validating an application

You should see a progress indicator appear briefly and then, if validation completes successfully, you should not see anything. To verify, check the Task pane at the bottom of the page. It should be empty after successful validation as shown in Figure 4-83.

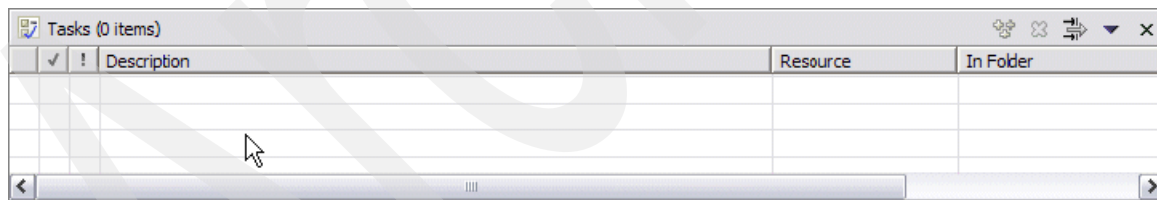


Figure 4-83 Empty task pane indicating a successful validation

Now, we examine what we would see if validation fails. First, we mess up our call flow a little bit to simulate some issues that are known to cause validation problems. For instance, delete the exit state for the audio item. This definitely causes a validation failure as shown in Figure 4-84.

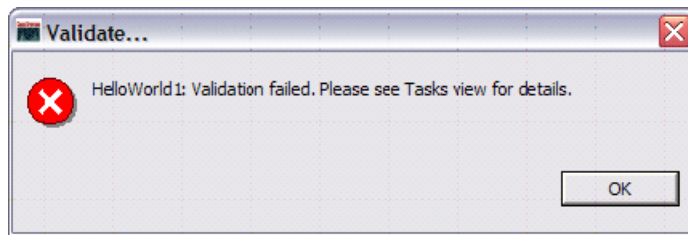


Figure 4-84 Validation failed message

**Note:** The presence of blue elements in your call flow are generally a good indication that not all required exit states have been connected properly as shown in Figure 4-85 on page 169.

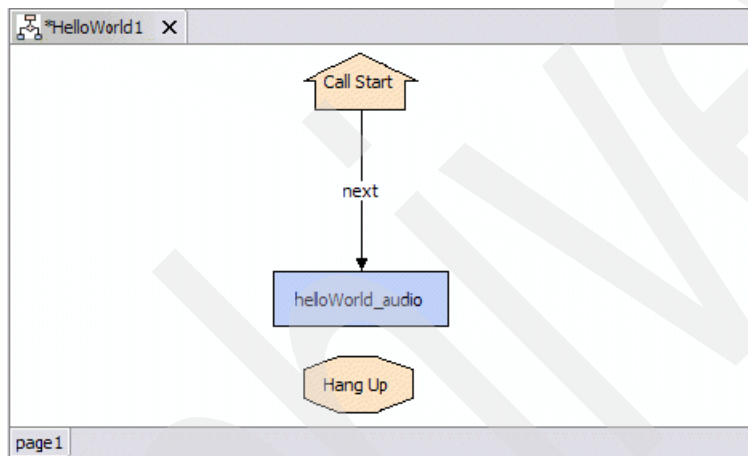


Figure 4-85 Blue element indicating that not all required exit states are connected

You can view the task pane for further details as shown in Figure 4-86.

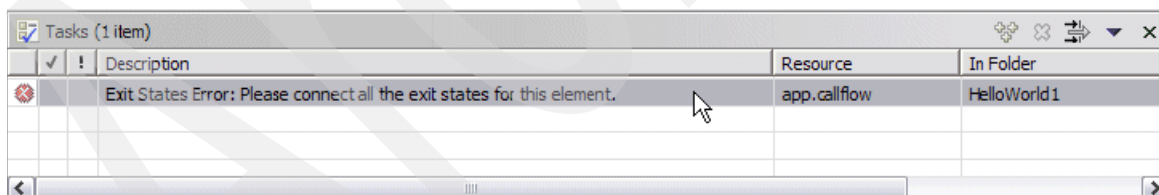


Figure 4-86 Task pane showing validation error message

Notice that the error description is Exit States Error: Please correct all the exit states for this element. One helpful feature of the task pane is that, if you click the entry, you are reading the offending item that will be selected in the call flow editor pane. This can be very helpful when you have a list of some 30 validation errors.

Now reconnect the exit state for the audio item to the hang up element, save the changes, revalidate the application, and everything should work properly.

Before proceeding further, we take a quick look at some very common validation errors and how they can be resolved.

### Common validation errors and how to correct them

Table 4-3 on page 170 shows a listing of the most common validation errors you will encounter while developing voice applications in studio. At this point, some of these error messages and proposed remedies will not be terribly meaningful for you. However, take note of this table. It might come in handy in the future. Also note that we have omitted some of the more arcane validation errors you might encounter because they occur only very rarely or their fixes are not so straightforward as to be amenable to explanation in a tabular format.

Table 4-3 Common Validation Errors and Remedies

Element Type	Error Message	Cause	Remedy
Decision	Exit States Error: Please connect all the exit states for this element.	Not all the exit states for this decision element have been connected.	Make sure that all exit states have been connected appropriately.
Decision	Exit States Error: Please define at least two exit states for this element.	A decision element must define at least two exit states to be a valid (Boolean) decision.	Create at least two exit states in the decisions General configuration
Decision	Configuration error: Please enter a valid class name for the source.	If you have a decision whose processing is performed by a Java class you must specify the class name ("Class" is selected in the drop down menu)	Define the full class name (including package name) for the class that provides the decision making functionality
Decision	Configuration error: Please enter a URI for the source.	If you have a decision whose processing is performed by the XML API delivered by a web page you must specify the page URI ("URI" is selected in the drop down menu)	Provide an absolute URI that includes a fully qualified domain name to the page that will provide the needed configuration XML
Decision	Configuration error: Please create the XML for this decision.	If you have a decision whose processing is performed by the XML API created in the XML editor pane ("XML" is selected in the drop down menu)	Click on the "Edit XML" button and define the appropriate decision XML in the editor pane that appears. This pane for the most part will only accept a well-formed XML string, but may sometimes accept XML that contains logical errors. Be aware of this if you are experiencing run time errors after successful validation.
Action	Exit States Error: Please connect all the exit states for this element.	Not all the exit states for this decision element have been connected.	Make sure that all exit states have been connected appropriately.

Element Type	Error Message	Cause	Remedy
Action	Configuration error: Please enter a valid class name for the source.	If you have a action whose processing is performed by a Java class you must specify the class name ("Class" is selected in the drop down menu)	Define the full class name (including package name) for the class that provides the desired functionality
Action	Configuration error: Please enter a URI for the source.	If you have an action whose processing is performed by the XML API delivered by a web page you must specify the page URI ("URI" is selected in the drop down menu)	Provide an absolute URI that includes a fully qualified domain name to the page that will provide the needed configuration XML
Voice element (includes Audio, Forms, etc...)	Configuration error: No URI or TTS content was specified for the audio item named "audio item ..." in the audio group "..." (count ...).	This error occurs when you forget to provide some kind of audio content (be it TTS or a URI pointing to prerecorded audio) for a defined audio group.	In this error message: "Configuration error: No URI or TTS content was specified for the audio item named "audio item 1" in the audio group "Initial" (count 1)." We can see that we have not provided a TTs string or an audio URI for the first defined audio item (indicated by "Count 1") in the audio group called "Initial". At minimum we must provide a TTS string (you may use a single space " " if you want to say nothing), a substitution tag that points to a TTS string or an audio URI.
Voice element (includes Audio, Forms, etc...)	Configuration error: The data value specified in the Say It Smart audio item named "audio item 1" in the audio group "Initial" (count 1) cannot be left blank or contain the following characters: <, >, ", ', and &.	When using Say It Smart plug-ins to read back audio content to the caller appropriate data must be supplied in the Say It Smart configuration.	Fixing these errors can be somewhat more tricky than errors with standard audio groups, but the error message is usually pretty accurate in terms of indicating where the problem lies. It often has to do with lack of a substitution string specifying the TTS content or the inclusion of illegal characters or incorrect formatting in said input.

## Application deployment

After an application has been thoroughly validated, it is finally ready for deployment and testing. Deployment is a process analogous to compilation in traditional programming languages. Well, actually it is a compilation. Studio takes generates a Web ARchive (WAR) file from the project, which is essentially an archive file consisting of: compiled Java classes and various static XML element configuration files and associated application resources that will be deployed to the CVP server to generate the VoiceXML code run in the voice browser on the gateway. There are some differences in the methods used for deployment of applications locally and remotely and also differences that will arise when you deploy an application in a lab or testing environment versus when you deploy them in a full production environment. Deployment of an application from studio is a very simple process, basically just a matter of two points and clicks. After this is done, there are a couple of additional steps that you must take to assure that the application is up and running on the Cisco CVP server and also that the gateway is correctly configured so that callers can actually access your application after it is deployed. We examine these scenarios in closer detail.

### ***Configuring Cisco CVP server for local deployments***

The term *Local Deploy* refers to the fact that you will be deploying your application from studio to a Cisco CVP server running on the same machine as studio. This is an eminently practical setup for development and testing and is by far the least complex of the possible deployment scenarios. After you have validated your application, all you need do is the following:

1. Right-click the project folder in the navigation pane and select **Deploy** as shown in Figure 4-87.

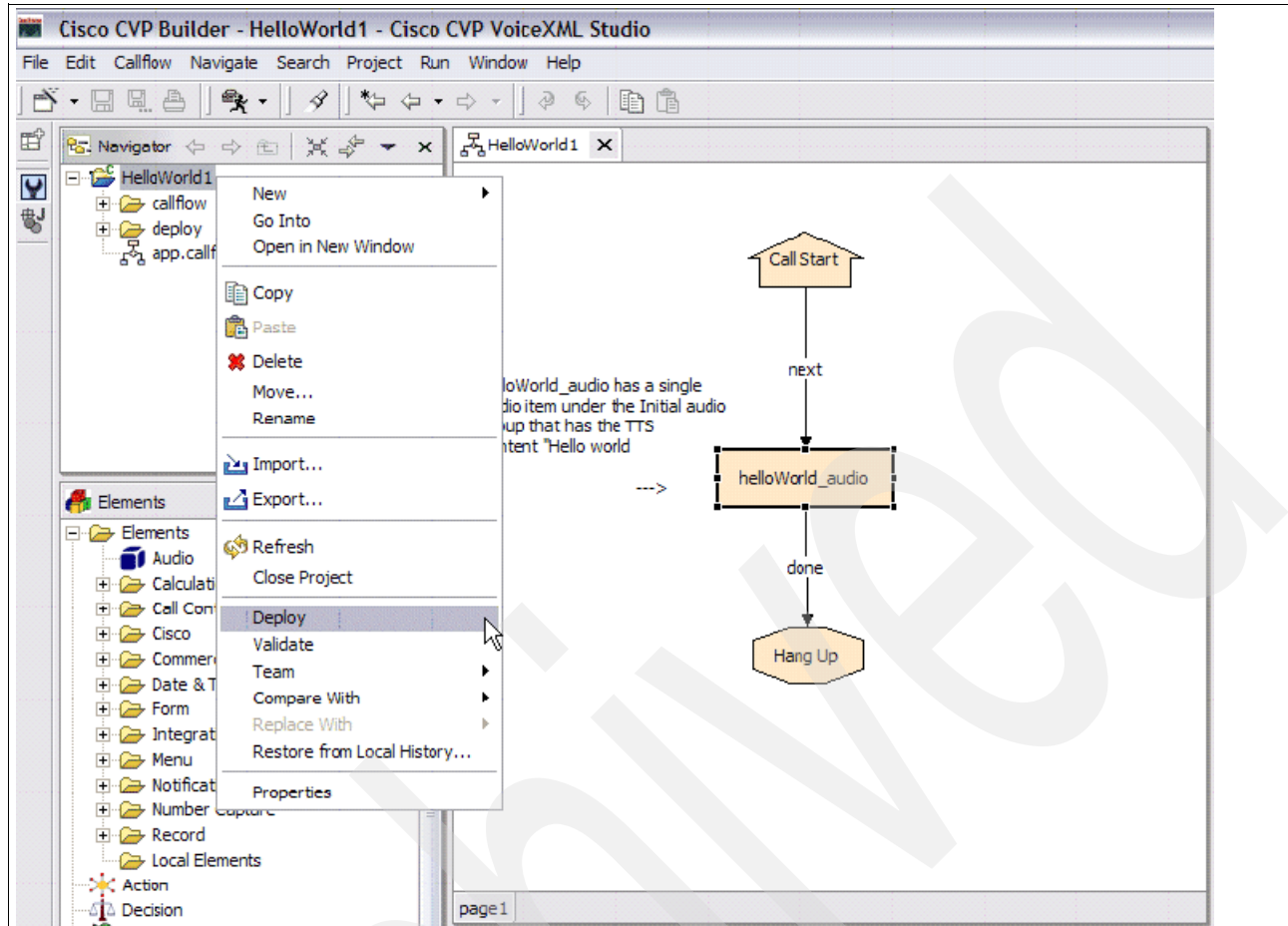


Figure 4-87 Deploying an application

2. Make sure the **Deploy Remotely** check box is not selected (see Figure 4-88 on page 174) and that the path to the Cisco CVP Server home directory is correct (if the server is installed in the default location then this should be C:\Cisco\CVP\Server which is the default value in the text box).

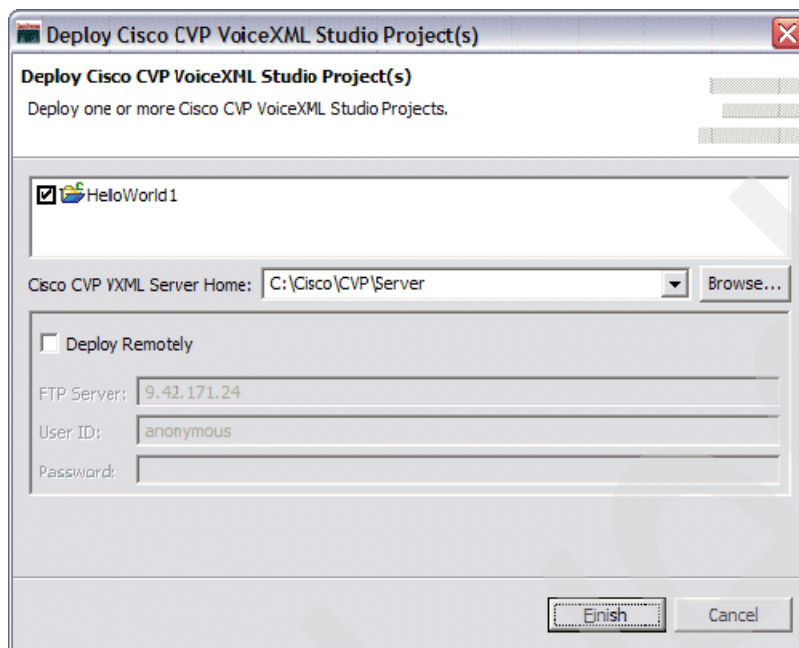


Figure 4-88 Deploy Cisco CVP VoiceXML Studio Project(s)

3. Click **Finish**. The application begins deploying as shown in Figure 4-89 on page 174.

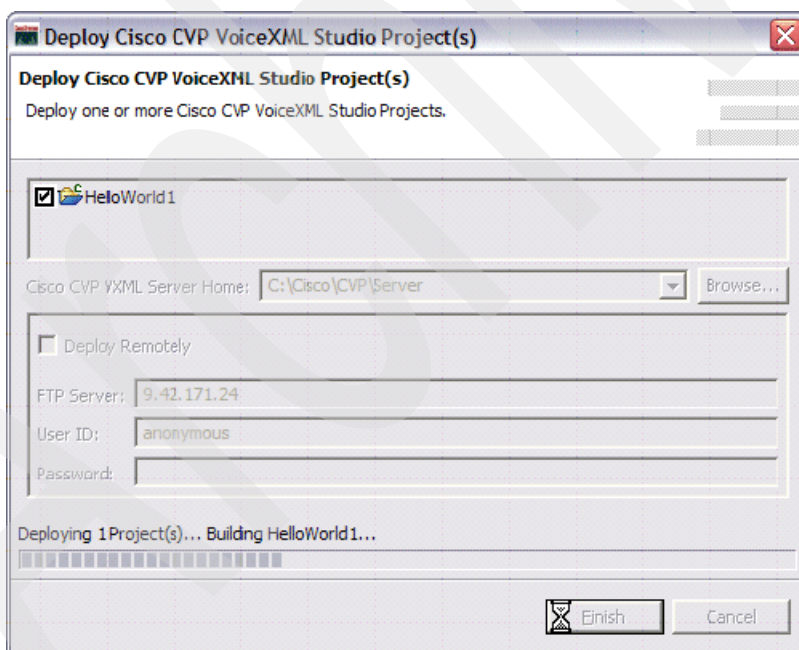


Figure 4-89 Deploying the project

Your application has now been successfully deployed. What has actually happened? Shortly we take a look at how a deployed application is structured.

### **Configuring Cisco CVP server for remote deployments**

In some cases, the Cisco CVP application server will not reside on the same development machine on which you run studio. In this case, you will have to deploy your applications remotely. The remote application deployment process in Studio is trivially easy with very little



difference from a local deploy. However, there are some important steps that you must take to configure your Cisco CVP server to enable remote deployments from another machine. These are as follows:

1. Configure FTP server (required)

In order to be able to deploy applications from a remote PC, an FTP server must be running on the Cisco CVP application server. A full discussion of FTP server setup and configuration is well outside the scope of this document. However, we do give you some pointers for Windows based systems on how to properly setup your FTP server.

- a. Server should listen on TCP port 21.
- b. Server should allow anonymous access (note this may be a serious security risk and is recommended only in a secure lab environment. Make sure you have a thorough understanding of your organizations security policy when setting up an FTP server on a production server).
- c. FTP home directory should point to the Cisco CVP server root (default on a Windows system is C:\Cisco\CVP\Server), home directory must allow full permissions (read, write, execute) for the designated Cisco CVP deployment user (in this case anonymous)

2. Configure a Telnet or SSH server (optional)

When deploying applications remotely, often you must be able to run various administrative scripts on the remote Cisco CVP server from your development station. If you are far away from the Cisco CVP server or the server is not physically accessible from your location, you will need some kind of remote command line access to the box in order to do so. The quick and dirty way to accomplish this is to enable the Telnet service (in Windows). You might want speak to your administrator before enabling a telnet server on a production server. A more secure option is to install an SSH server on the CVP server. We leave the implementation details to the user, but believe us, it is incredibly handy not to have to run over to the server and key in commands every time you have to test an updated version of your application.

3. Enable Some Form of Remote Control (optional)

An alternative to enabling remote command line access to your Cisco CVP application server is to use some kind of remote control software (e.g. Remote Desktop Connection on Windows boxes). This can come in very handy and offers somewhat more flexibility than simple command line connectivity to the server. It can be considered overkill, however, if you only need to be able to run the administrative scripts from your development station. But we have found that having remote control over the application server can come in extremely handy. Once again we implore you to thoroughly discuss the security implications of installing remote control software on the server in question.

4. When your Cisco CVP server has been configured appropriately, all you need do is right-click your project folder in the navigation pane in studio and select **deploy**. You then see the familiar deployment dialog box as shown in Figure 4-90.

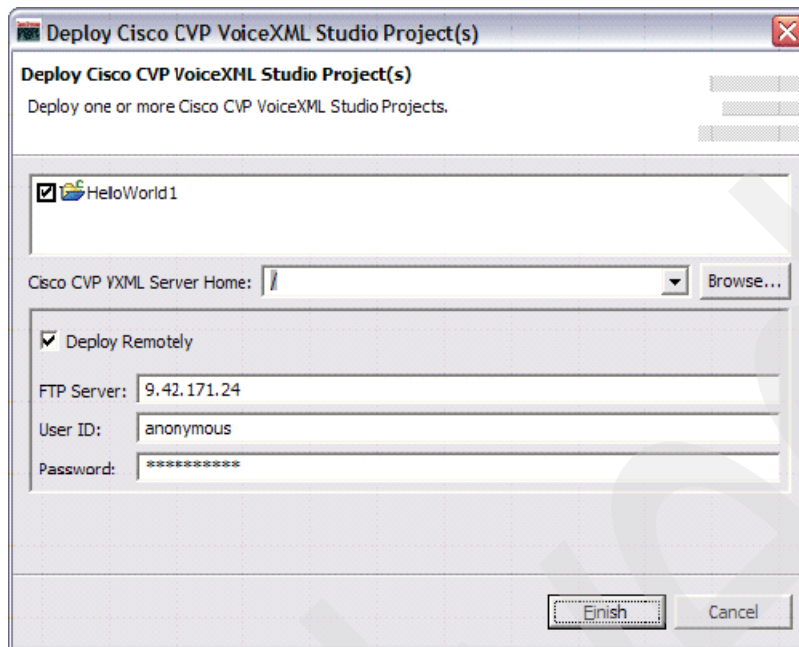


Figure 4-90 Deploying Remotely

5. Ensure that the following are true:
  - **Deploy Remotely** check box is selected.
  - The FTP Server field contains the IP address of your remote Cisco CVP application server.
  - The User ID contains the user name of a user who has full permissions to the application server root directory (as mentioned in the server configuration section, we feel comfortable using the anonymous user in our lab environment) and the Password field must be populated with the FTP users password (for an anonymous user this can be any old garbage, hence the beauty of the anonymous user you do not have to key in an actual password every time you deploy).
  - Finally ensure that the Cisco CVP VoiceXML Server Home points to the application server root (as seen by the FTP server, if you configured your FTP server as we recommended above it should simply be the FTP server root “/”).
6. After all the necessary information has been provided, simply click **Finish** and your application will be deployed to the remote server in exactly the same fashion as it was when you deployed it to your local server.

### ***Directory structure of deployed applications***

Here, we briefly examine the structure of a deployed CVP voice application as shown in Figure 4-91.

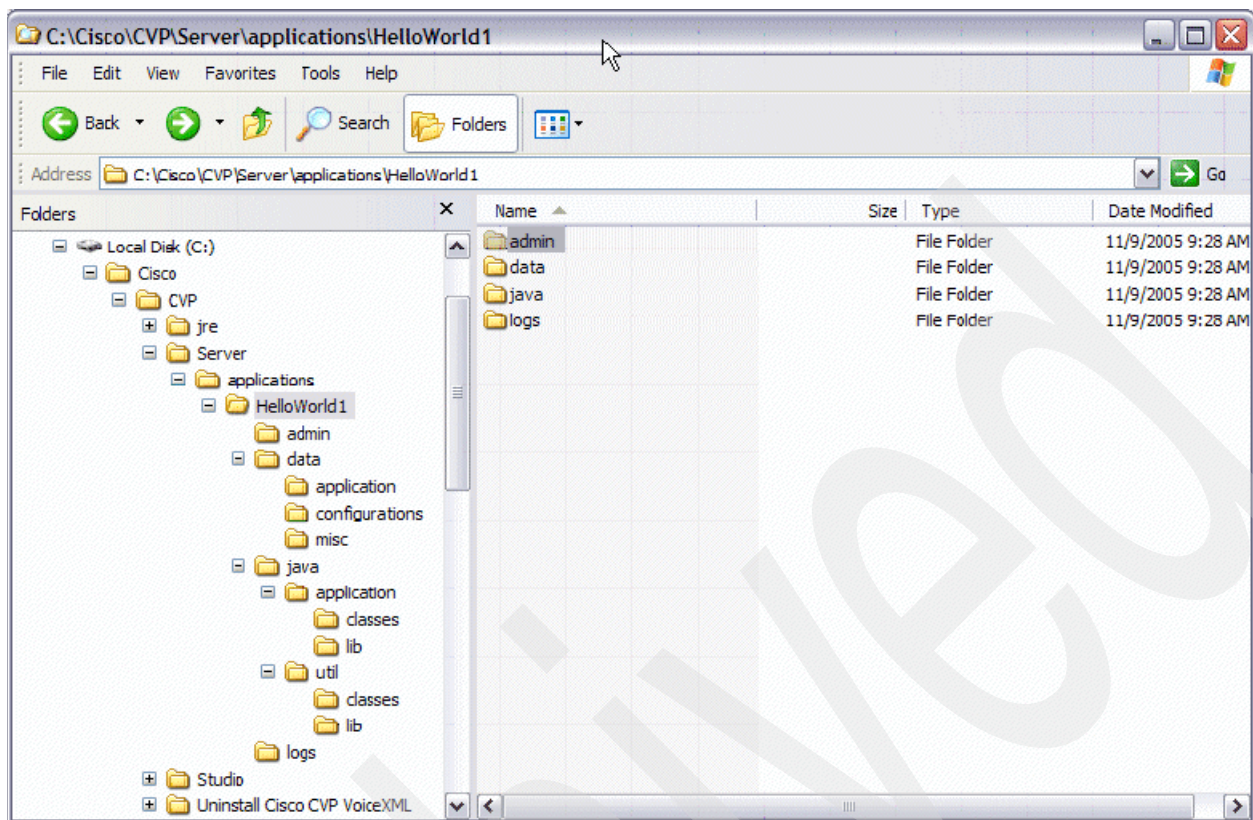


Figure 4-91 Directory structure of a deployed Cisco CVP voice application

Screen shot reprinted by permission from Microsoft Corporation

As you can see, the deployment process results in the creation of a fairly elaborate directory structure with a root directory that has the same name as the deployed application (in this case HelloWorld1) containing a number of interesting files under the Applications directory of the server root. We take a quick look at the contents of some directories that are of concern to the developer.

- The Admin directory (see Figure 4-92 on page 178) contains scripts related to the graceful administration of deployed applications. Occasionally, you will want perform administrative functions such as updating, suspending, releasing, and so on to the application. These functions can all be performed gracefully and painlessly by running these scripts (the \*.bat files are batch files for use on Windows platforms). For now, we defer a lengthy discussion of the various administrative scripts for a bit. We will come back to this topic shortly.

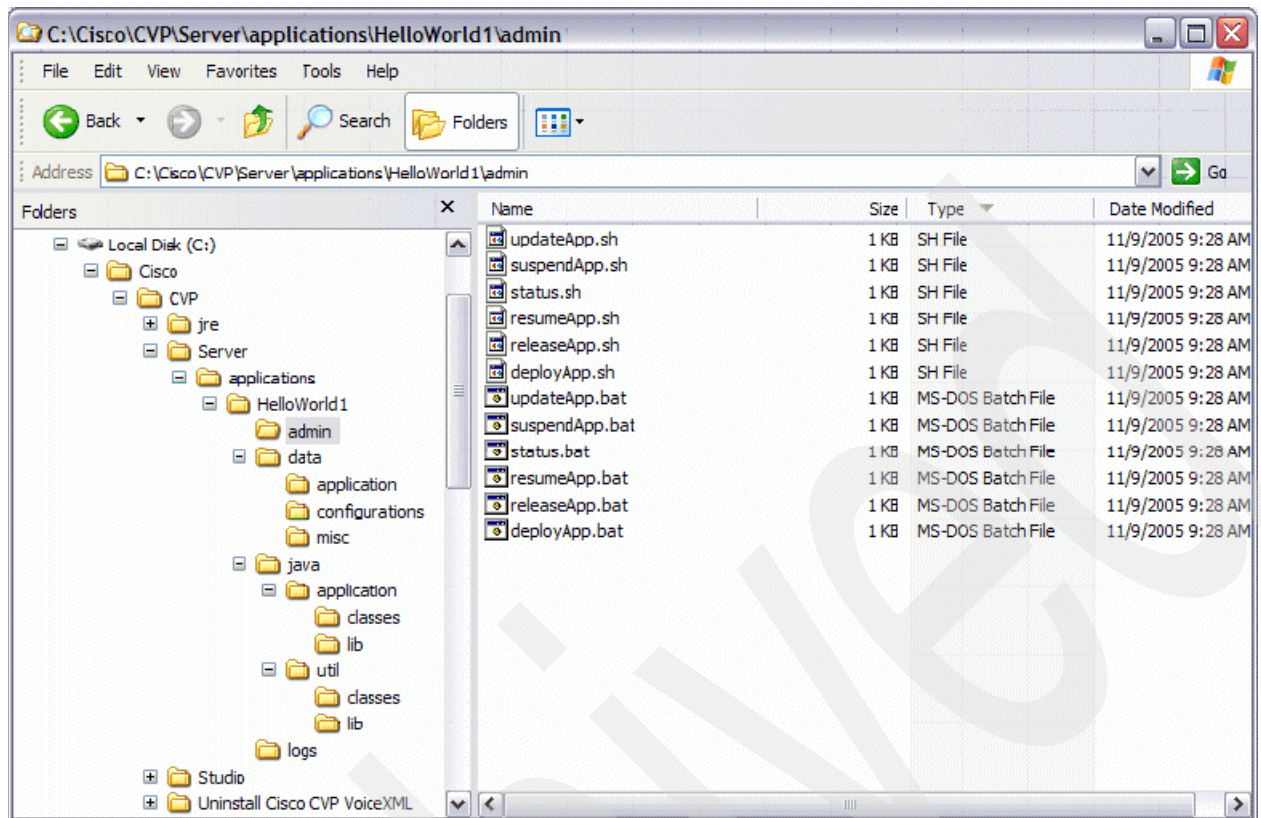


Figure 4-92 Directory structure of a deployed Cisco CVP voice application - admin directory

Screen shot reprinted by permission from Microsoft Corporation

- The Logs directory (see Figure 4-93 on page 179) is of vital importance. This is where you can find various administrative, application, and error logs after your application is deployed and running. When you first deploy your application, you should only see one file in the logs directory and that is the admin\_history log (see Figure 4-94 on page 179) for the date you deployed the application. This log is a very simple one that contains a history of any administrative actions taken on that application on the specified date.



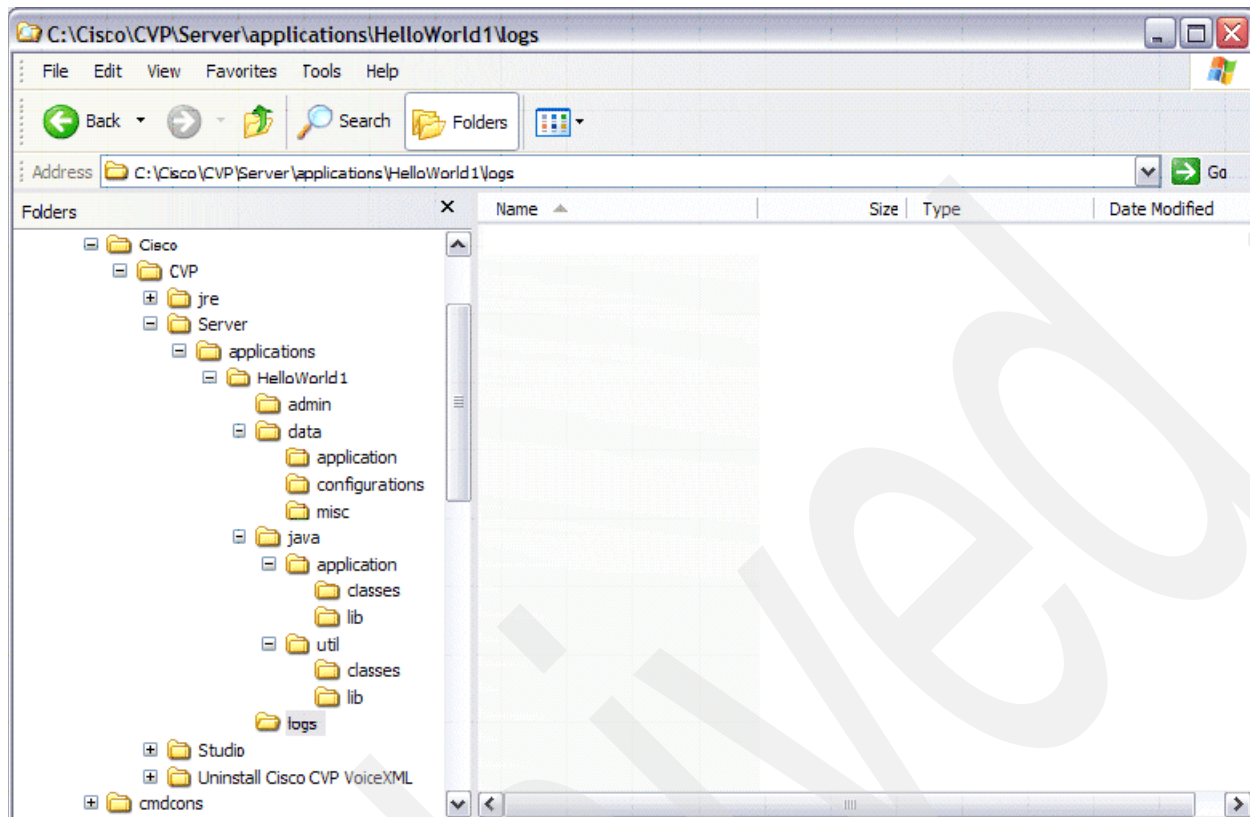


Figure 4-93 Directory structure of a deployed CVP voice application - logs directory  
Screen shot reprinted by permission from Microsoft Corporation



Figure 4-94 admin history log

We discuss the plethora of information a developer can cull from a thorough examination of various log files when we discuss troubleshooting and application debugging.

- ▶ The only other directories that are of any direct concern to the developer are the Java directory and its associated subdirectories. This is the location where additionally class and library files that might be used in more advanced applications are deployed. For now, this is not a concern. You will have also noticed that various other directories contain an assortment of XML files. These are simply application and static element configuration files, feel free to take a peek at their contents if you are so inclined.

### **Administrative scripts and graceful application manipulation**

The administrative scripts (see Table 4-4 on page 180) used on the Cisco CVP application server are used to gracefully modify applications as required. The functionality of these scripts includes the ability to perform the following actions on applications:

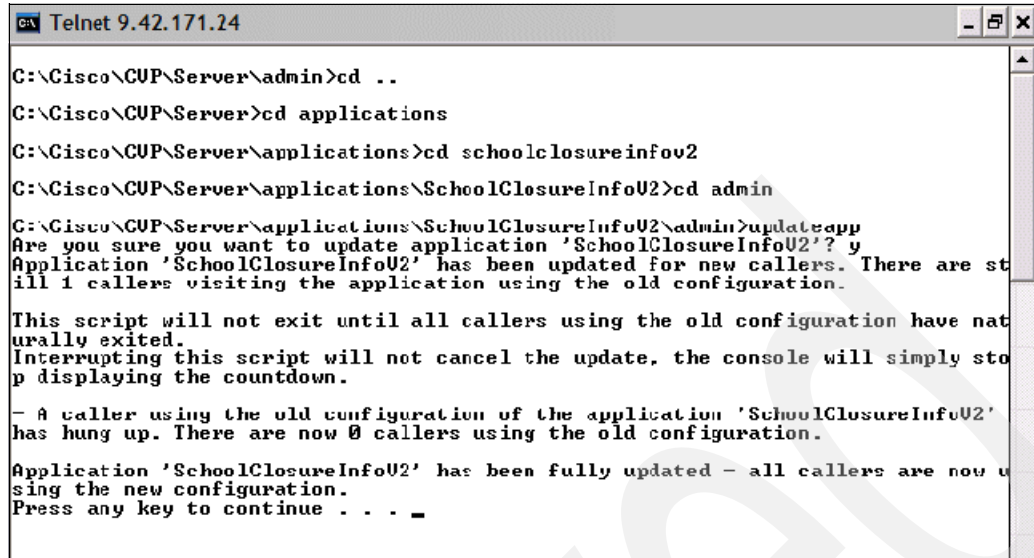
- ▶ Delete
- ▶ Update
- ▶ Deploy
- ▶ Suspend

- ▶ Resume
- ▶ Release

These scripts allow administrative actions to be accomplished gracefully (that is to say that existing callers continue to experience the application as it existed before the execution of the script in question, while new callers will experience the application as it exists after the execution of the script). Administrative scripts can also be executed globally (affecting all applications deployed on the server) or locally (affecting only a single application).

Table 4-4 Administrative scripts and their description

Administrative Script Name	Description
<b>Application-Level Scripts</b>	
suspendApp	Suspends the application in which this script resides.
resumeApp	Resumes the application in which this script resides.
deployApp	Prompts CVP VoiceXML Server to load the application in which this script resides (does nothing if the application is already deployed).
updateApp	Prompts the Server to reload into memory the configuration of the application in which this script resides.
releaseApp	Prompts the Server to remove from memory the application in which this script resides so that its folder can be deleted.
status	Displays current information about the application in which this script resides.
<b>Server Scripts</b>	
suspendServer	Suspends all applications deployed on the Server.
resumeServer	Restores the status of each application to what they were at the time the Server was suspended.
deployAllNewApps	All applications deployed to the Server since the last time the application server started up or the deployAllNewApps script was called are now loaded into memory and can handle calls.
flushAllOldApps	When called, all applications in the Server whose folders were deleted are removed from memory.
updateAllApps	Prompts each application deployed on the Server to load its configuration from scratch from the application files.
updateCommonClasses	Reloads all classes deployed in the common directory of the Server.
status	Displays current information about the Server and all the applications deployed on it.
getVersions	Displays the version number of all CVP VoiceXML components installed on the Server. The location in which the WARFileName file is located must be passed as an argument.
importLogs	Imports the Cisco CVP VoiceXML activity logs of an application into a database.

A screenshot of a Telnet window titled 'Telnet 9.42.171.24'. The window shows a series of directory navigation commands and the execution of the 'updateapp' command. The output of 'updateapp' includes a confirmation prompt, a confirmation of the update, a warning about callers using the old configuration, and a final confirmation that the application is fully updated.

```
C:\Cisco\CVP\Server\admin>cd ..
C:\Cisco\CVP\Server>cd applications
C:\Cisco\CVP\Server\applications>cd schoolclosureinfov2
C:\Cisco\CVP\Server\applications\SchoolClosureInfoV2>cd admin
C:\Cisco\CVP\Server\applications\SchoolClosureInfoV2\admin>updateapp
Are you sure you want to update application 'SchoolClosureInfoV2'? y
Application 'SchoolClosureInfoV2' has been updated for new callers. There are still 1 callers visiting the application using the old configuration.

This script will not exit until all callers using the old configuration have naturally exited.
Interrupting this script will not cancel the update, the console will simply stop displaying the countdown.

- A caller using the old configuration of the application 'SchoolClosureInfoV2' has hung up. There are now 0 callers using the old configuration.

Application 'SchoolClosureInfoV2' has been fully updated - all callers are now using the new configuration.
Press any key to continue . . . -
```

Figure 4-95 Administering an application using the updateApp command

► deployApp

Occasionally an application must be deployed explicitly using an administrative script. This can be accomplished by navigating to the application admin directory and then running the appropriate script for the server platform. For example, if a developer wanted to force deployment of the HelloWorld1 application deployed on a Windows server™, then he would do the following (assuming server root is in default location):

```
C:\Cisco\CVP\Server\applications\HelloWorld1\admin\deployApp.bat
```

Alternately, the global version of the script could be run to deploy all new applications deployed on the server:

```
C:\Cisco\CVP\Server\admin\deployAllNewApps.bat
```

**Troubleshooting:** Sometimes after deploying an application to a remote server from studio, the application does not come into service. You will know this is happening because when you attempt to access an application you are sure has no problems in it you hear the generic error message played. In these cases, explicitly running the deployapp script from the application's admin directory might resolve the problem.

► updateApp

Occasionally an application must be updated. This will happen very frequently as you test various iterations of your application during the development process (much less often in the production phase). When you redeploy an application, you should run the updateapp script. This can be accomplished by navigating to the application admin directory and then running the appropriate script for the server platform. For example, if a developer wanted to update the HelloWorld1 application deployed on a Windows server, then he would do the following (assuming server root is in default location):

```
C:\Cisco\CVP\Server\applications\HelloWorld1\admin\updateapp.bat
```

Alternately, the global version of the script could be run to update all applications deployed on the server:

```
C:\Cisco\CVP\Server\admin\updateallapps.bat
```

**Troubleshooting Note:** If while developing you have redeployed your application but do not notice your changes, make sure to rerun the updateapp script to force an update.

► **suspendApp**

Occasionally an application must be temporarily suspended. Callers that attempt to access a suspended application will be played the application suspended message specified in the project settings in studio. Suspension of an application can be accomplished by navigating to the application admin directory and then running the appropriate script for the server platform. For example, if a developer wanted to suspend the HelloWorld1 application deployed on a Windows server, then he would do the following (assuming server root is in default location):

```
C:\Cisco\CVP\Server\applications\HelloWorld1\admin\suspendapp.bat
```

Alternately, the global version of the script could be run to suspend all applications deployed on the server:

```
C:\Cisco\CVP\Server\admin>suspendServer.bat
```

► **resumeApp**

The resumeapp script is run in order to bring a temporarily suspended application back into service. For example, if a developer wanted to resume the HelloWorld1 application deployed on a Windows server after temporarily suspending it, then he would do the following (assuming server root is in default location):

```
C:\Cisco\CVP\Server\applications\HelloWorld1\admin\resumeApp.bat
```

Alternately, the global version of the script could be run to resume all suspended applications deployed on the server:

```
C:\Cisco\CVP\Server\admin>resumeServer.bat
```

► **releaseApp**

The releaseApp script is run in order to completely release application resources from server memory so that it can be deleted. Attempts to delete an application directory from the server while it is running will fail. The application must first be released then deleted. For example if a developer wanted to release the HelloWorld1 application deployed on a Windows server after temporarily suspending it then he would do the following (assuming server root is in default location):

```
C:\Cisco\CVP\Server\applications\HelloWorld1\admin\releaseApp.bat
```

The HelloWorld1 directory can now be deleted successfully from the server application directory. The global version of the script can be run to remove all individually released applications residing on the server:

```
C:\Cisco\CVP\Server\admin>flushAllOldApps.bat
```

**Using the scripts (examples)**

Figure 4-96 on page 183 shows an example of running an administrative script on a specific application.



```

Telnet 9.42.171.24
Volume Serial Number is 703A-8103

Directory of C:\Cisco\CVP\Server\applications\HelloWorld1\admin

11/09/2005 09:52a <DIR> .
11/09/2005 09:52a <DIR> ..
11/09/2005 10:41a 330 deployApp.bat
11/09/2005 10:41a 218 deployApp.sh
11/09/2005 10:41a 333 releaseApp.bat
11/09/2005 10:41a 221 releaseApp.sh
11/09/2005 10:41a 330 resumeApp.bat
11/09/2005 10:41a 219 resumeApp.sh
11/09/2005 10:41a 330 status.bat
11/09/2005 10:41a 218 status.sh
11/09/2005 10:41a 334 suspendApp.bat
11/09/2005 10:41a 222 suspendApp.sh
11/09/2005 10:41a 333 updateApp.bat
11/09/2005 10:41a 222 updateApp.sh
12 File(s) 3,310 bytes
2 Dir(s) 17,704,333,312 bytes free

C:\Cisco\CVP\Server\applications\HelloWorld1\admin>updateapp
Are you sure you want to update application 'HelloWorld1'? y
Application 'HelloWorld1' has been fully updated.
Press any key to continue . . .

```

Figure 4-96 updateapp command

Figure 4-97 shows an example of running one of the global administrative scripts.

```

Telnet 9.42.171.24

08/17/2005 03:33p 199 resumeServer.sh
08/17/2005 03:33p 303 status.bat
08/17/2005 03:33p 191 status.sh
08/17/2005 03:33p 310 suspendServer.bat
08/17/2005 03:33p 198 suspendServer.sh
08/17/2005 03:33p 307 updateAllapps.bat
08/17/2005 03:33p 195 updateAllapps.sh
08/17/2005 03:33p 313 updateCommonClasses.bat
08/17/2005 03:33p 201 updateCommonClasses.sh
20 File(s) 33,453 bytes
3 Dir(s) 17,704,333,312 bytes free

C:\Cisco\CVP\Server\admin>updateallapps
Are you sure you want to update all applications? y
- Application 'DIMFTest1' has been updated for all callers.
- Application 'HelloWorld' has been updated for all callers.
- Application 'HelloWorld1' has been updated for all callers.
- Application 'SchoolClosureInfo' has been updated for all callers.
- Application 'SchoolClosureInfo02' has been updated for all callers.
- Application 'SchoolConfirmTest' has been updated for all callers.
- Application 'TestInlineGrammar' has been updated for all callers.

To check if all callers to a particular application are using the new configurat
ion, use the 'status' script in that application's admin folder.
Press any key to continue . . .

```

Figure 4-97 updateallapps command

## Tips for working efficiently in the development environment

In this section, we provide a few tips to enable you to work more efficiently in your development environment.

### Placement of application resources

When creating complex applications that reference external resources such as audio files and grammars, it is often very useful to have all those resources in a well-known location on the application server. This is a practice we attempt to adhere to as much as possible to avoid the irritation of dealing with error.badfetch and similar problems. A very convenient way to configure your application server is as follows. On a windows installation of Cisco CVP server the physical directory "C:\Cisco\CVP\Tomcat 4.1\webapps\CVP" maps to the following URL:

<http://localhost:8080/CVP>

This creates a directory structure similar to Figure 4-98 that will allow you to very simply reference external resources from within your application.

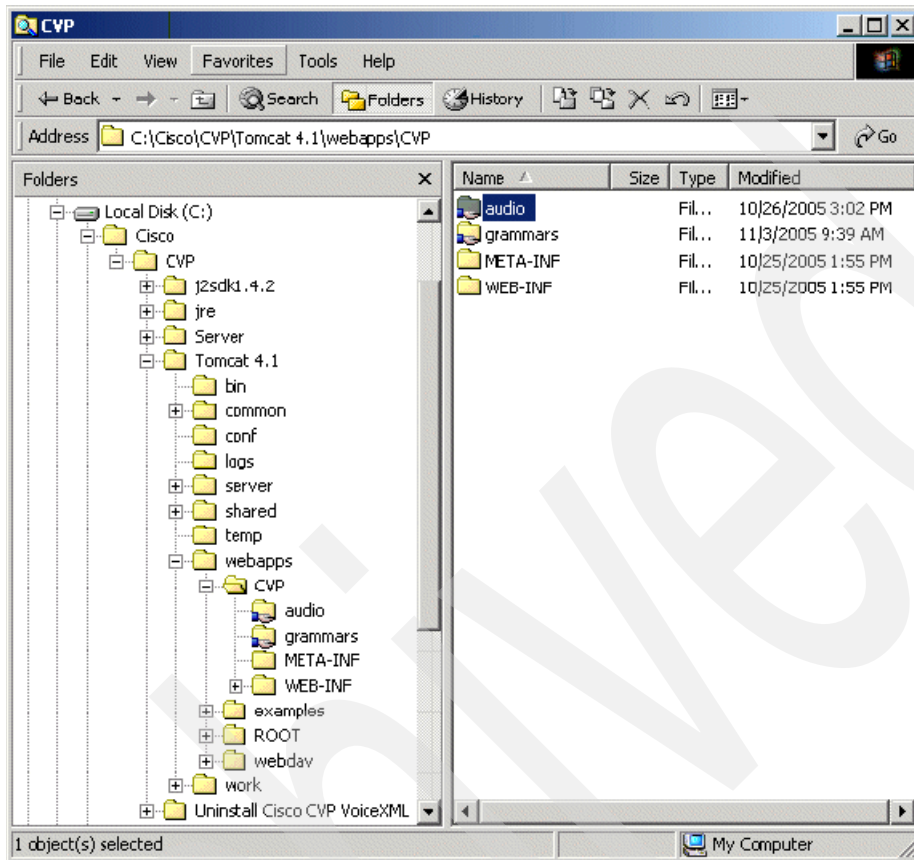


Figure 4-98 Directory structure for referencing external resources from within your application  
Screen shot reprinted with permission from Microsoft Corporation

This is just to give you an idea of a directory structure that is amenable to convenient deployment and manipulation of external application resources. We have created an audio folder where we place various directories named after our applications that contain prerecorded audio files for use in the application (we then simply set the default audio path to /CVP/audio/application-name in studio, audio resources can then simply be referenced by file name and do the same thing for grammars we create a grammars root with subdirectories named after deployed applications and then, in studio we can simply reference our external grammars in the following way:

```
http://localhost:8080/CVP/grammars/application-name/grammar-file-name.grxml
```

We also share these folders so that we can create mapped network drives to these folders on our development stations so that resources can be manipulated easily. Alternately, you can add these folders to your FTP server and manipulate these resources via FTP. On Windows, simply create a new virtual directory in the FTP portion of the IIS management snap-in. We find this type of setup to be quite efficient and saves a lot of time. Of course, there are significant security implications to this type of setup and we recommend it be used only in a secured lab environment.

### Starting the server from the command line

When developing Java classes for custom components, standard actions, and so on, System.out.println() debugging methodology can be very productive to be able to view standard output from the Cisco CVP application server when your class runs for debugging

and diagnostic purposes (a very handy way to debug Java classes deployed to the Cisco CVP server). A very nice way to do this is as follows:

1. If you installed Cisco CVP server as a service and the service is started, you must first stop the service. You can do this in two ways:
  - a. From the command line: You can stop the Cisco CVP application server service from the command line by issuing the following command:
 

```
net stop "Apache Tomcat 4.1 for CVP VoiceXML Server"
```
  - b. Using the Services Administrative Tools snap-in (see Figure 4-99).

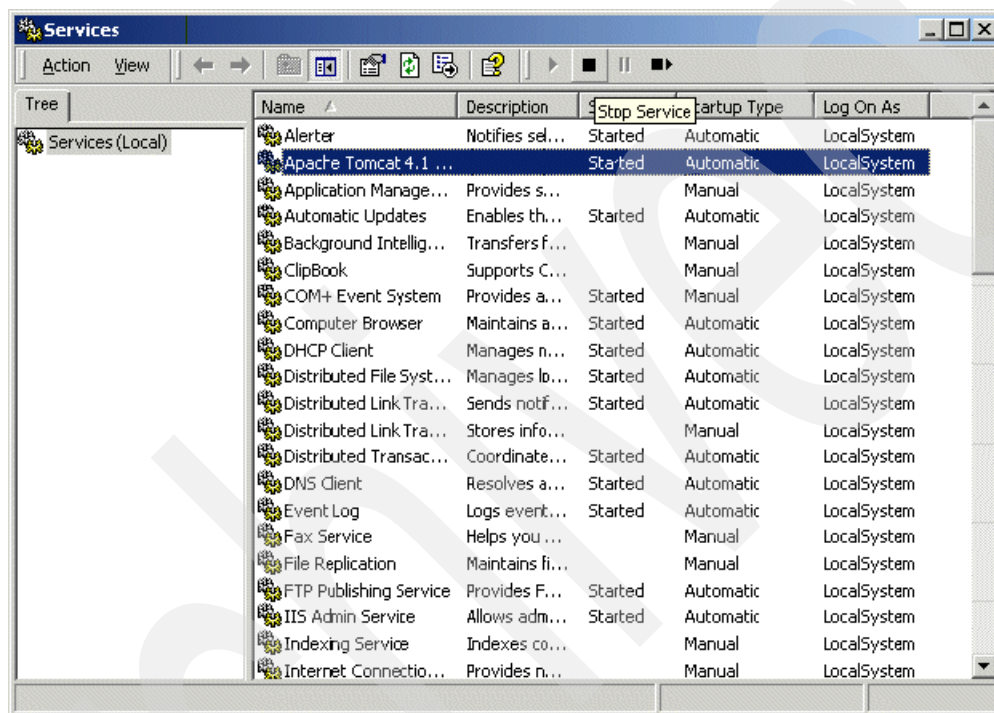


Figure 4-99 Services Administrative Tool snap-in  
Screen shot reprinted with permission from Microsoft Corporation

2. Create a text file to capture the standard output.

**Note:** It is even more handy if you create this text file in a shared drive that you can access from your development station, then all you need to do is open the file in a text editor that updates files dynamically on your desktop and you can see standard output from the server in real time! TextPad is our personal favorite. You would generally call this file stdout\_log.txt or something similar.

3. Start the server from the command line and redirect standard output to the text file you just created:
 

```
catalina run > g:\stdout_log.txt
```
4. Open the stdout\_log.txt file in Textpad on your desktop. This is a very handy way to debug Java classes deployed to the Cisco CVP server.

### Sharing log directories

It can also be very helpful to share application log directories so that they can be accessed from your desktop. Then you can open the log file once in TextPad. Whenever a log entry is

written by the server, the display will be refreshed. There is no need to constantly refresh or reopen log files to view new entries. Our experience is that you might sometimes have to view the logs dozens of times when attempting to debug even simple problems, so this is an enormous time saver.

## Testing your application

We have reached the point where we have deployed our application successfully from studio to our Cisco CVP application server and run the appropriate administration scripts to ensure that our application is deployed and fully up-to-date. All that remains now is to add a few lines of configuration to our Cisco gateway so that the embedded voice browser can find and run our application, then we can actually test our HelloWorld1 application.

### Testing with POTS

Follow these steps to begin testing.

#### 1. Preparing the Cisco gateway to run your application

Preparing the Cisco gateway to run your application is relatively simple (for further information about configuring Cisco gateways see Chapter 2, “Basic deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3” on page 13). All that needs be done is register the service and then add a POTS dial peer to handle incoming calls.

Example 4-1 shows a sample configuration (assuming the gateway is configured as specified in Chapter 2, “Basic deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3” on page 13) that will allow the gateway to access an English language application called HelloWorld9 residing on Cisco CVP servers 9.42.171.24 and 9.42.171.94 (two CVP servers clustered with a load balancer to provide redundancy) listening on port 8080 (default CVP HTTP port):

#### Example 4-1 Sample configuration

---

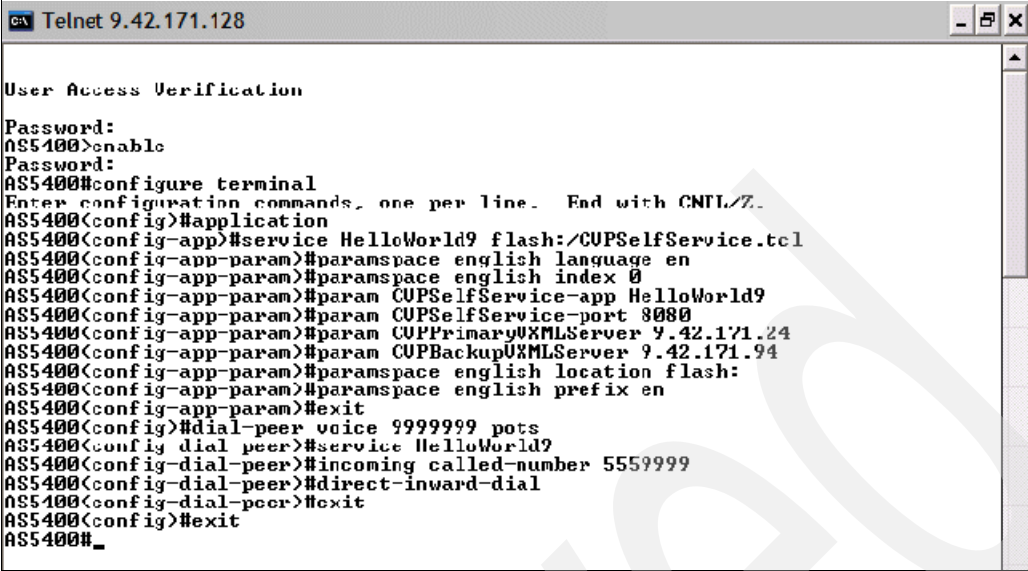
```
service HelloWorld9 flash:/CVPSelfService.tcl
  paramspace english language en
  paramspace english index 0
  param CVPBackupVXMLServer 9.42.171.94
  param CVPSelfService-app HelloWorld9
  param CVPSelfService-port 8080
  paramspace english location flash:
  param CVPPrimaryVXMLServer 9.42.171.24
  paramspace english prefix en
```

Here is the POTS dial peer that can be used to access the application (by dialing the number 555-9999)

```
dial-peer voice 9999999 pots
  service HelloWorld9
  incoming called-number 5559999
  direct-inward-dial
```

---

This configuration could be added to the running-config on the gateway as shown in Figure 4-100 on page 187.



```
Telnet 9.42.171.128

User Access Verification

Password:
AS5400>enable
Password:
AS5400#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
AS5400(config)#application
AS5400(config-app)#service HelloWorld9 flash:/CVPSelfService.tcl
AS5400(config-app-param)#paramspace english language en
AS5400(config-app-param)#paramspace english index 0
AS5400(config-app-param)#param CVPSelfService-app HelloWorld9
AS5400(config-app-param)#param CVPSelfService-port 8080
AS5400(config-app-param)#param CVPPrimaryVXMLServer 9.42.171.24
AS5400(config-app-param)#param CVPBackupVXMLServer 9.42.171.94
AS5400(config-app-param)#paramspace english location flash:
AS5400(config-app-param)#paramspace english prefix en
AS5400(config-app-param)#exit
AS5400(config)#dial-peer voice 9999999 pots
AS5400(config-dial-peer)#service HelloWorld9
AS5400(config-dial-peer)#incoming called-number 5559999
AS5400(config-dial-peer)#direct-inward-dial
AS5400(config-dial-peer)#exit
AS5400(config)#exit
AS5400#_
```

Figure 4-100 Adding a configuration to the running-config

To ensure that you have added your application correctly, issue the **show running-config** command from enable mode and ensure the information you entered is correct.

Finally, to make this configuration change permanent, save the changes to the startup-config (NVRAM) by issuing the **copy running-config startup-config** command from enable mode.

The process would be exactly the same for our HelloWorld1 application except the configurations would look something like Example 4-2.

#### Example 4-2 HelloWorld1: configuration

```
service HelloWorld1 flash:/CVPSelfService.tcl
paramspace english language en
paramspace english index 0
param CVPBackupVXMLServer 9.42.171.94
param CVPSelfService-app HelloWorld1
param CVPSelfService-port 8080
paramspace english location flash:
param CVPPrimaryVXMLServer 9.42.171.24
paramspace english prefix en
```

Here is the POTS dial peer that can be used to access the application (by dialing the number 555-9999)

```
dial-peer voice 9999999 pots
service HelloWorld1
incoming called-number 5559999
direct-inward-dial
```

## 2. Performing the test

You can now test the application by dialing in to the gateway with number 555-9999. You should here the TTS engine read you the text Hello World.

### Testing with H323 VoIP Softphone

Follow these steps to begin testing.

- Preparing the Cisco gateway to run your application

First ensure that your gateway is capable of handling H323 calls (refer to Chapter 2, “Basic deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3” on page 13 for further information about gateway configuration). The running-config for the gateway should contain an entry similar to Example 4-3 (this is on a Cisco AS5400HPX Universal Gateway).

*Example 4-3 Configuring a Cisco AS5400HPX Universal Gateway to handle H323 calls*

```
voice service voip
fax protocol t38 ls-redundancy 0 hs-redundancy 0 fallback cisco
h323
```

One of your fastethernet interfaces should be configured to accept H323 calls

```
interface FastEthernet0/0
ip address 9.42.171.128 255.255.255.0
no ip redirects
ip route-cache same-interface
no ip mroute-cache
duplex full
speed 100
no keepalive
no cdp enable
h323-gateway voip interface
h323-gateway voip id Gatekeeper-3660 ipaddr 9.42.171.186 1719
h323-gateway voip h323-id 9.42.171.128
h323-gateway voip tech-prefix 1#
```

On a Cisco 1751 gateway the H323 configuration might look something like the following:

```
voice service voip
allow-connections h323 to h323
fax protocol t38 ls-redundancy 0 hs-redundancy 0 fallback none
h323
```

When you are certain your gateway can accept H323 calls (and assuming that you have registered your service as we did above for the POTS testing configuration), all you need do is add a VoIP dial peer to handle incoming requests for the HelloWorld1 application as shown in Example 4-4.

*Example 4-4 Configuring a dial-peer on a Cisco AS5400HPX Universal Gateway*

```
dial-peer voice 999 voip
translation-profile incoming block
service helloworld1
incoming called-number 999
dtmf-relay rtp-nte h245-signal h245-alphanumeric
codec g711ulaw
no vad
```

- Configuring the Softphone

Configuring the Softphone is quite easy. All that you need to do is specify the gateway IP address and the address for the service you want to test. In this example, our gateway IP address is 9.42.171.128 and we have set up a dial peer that listens for the HelloWorld1 application on address 999.

- Performing the test

Assume that we are using the H323 OpenPhone software as our soft phone:

<http://www.openh323.org>

We can now attempt to place our test call as shown in Figure 4-101.

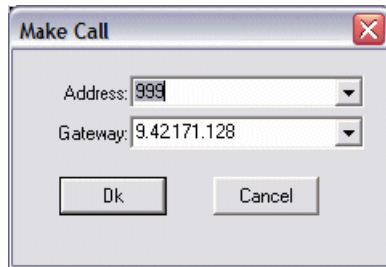


Figure 4-101 Making a call using OpenPhone

Our test call is now in progress as shown in Figure 4-102.

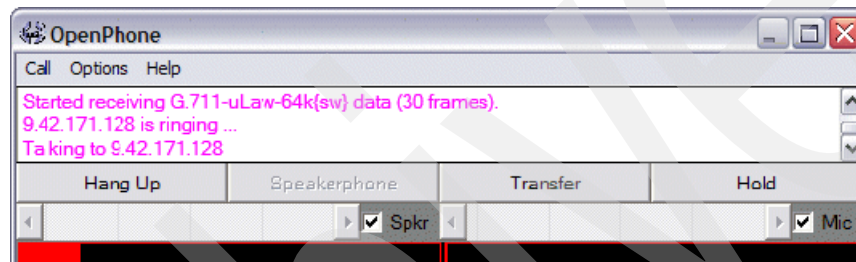


Figure 4-102 A test call in progress using OpenPhone

## HelloWorld2: hello world with prerecorded audio

We have now deployed and tested successfully HelloWorld1, which is a small sample application that reads the TTS string Hello world. Now we can create a new version of our venerable HelloWorld and modify it so that it instead of saying “Hello world” by reading a TTS string, it does same thing using prerecorded audio. Call the project HelloWorld2 and set the default audio path for the application to /CVP/audio/HelloWorld2/.

### Acceptable audio formats

This is a good time to mention sound formats that work properly on Cisco gateways:

- ▶ \*.au
- ▶ \*.wav

### Recording audio files

We can record the audio for our prompt using the Windows sound recorder. We simply record the phrase Hello world and save it in 8-bit, 8-kHz mono CCIT mu-Law WAV format as shown in Figure 4-103, Figure 4-104, and Figure 4-105.





Figure 4-103 Microsoft Windows Sound Recorder  
Screen shot reprinted by permission from Microsoft Corporation

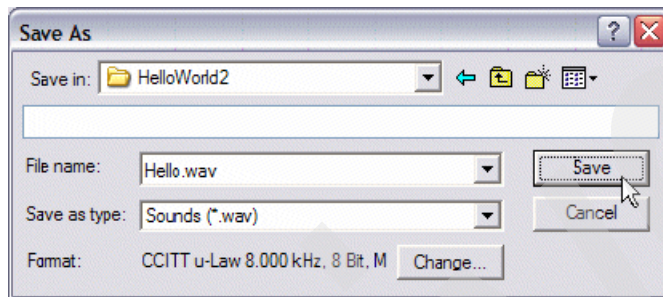


Figure 4-104 Microsoft Windows Sound Recorder: Save As dialog

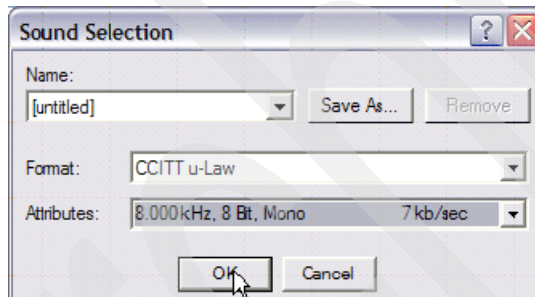


Figure 4-105 Microsoft Windows Sound Recorder - Change... button dialog

## Putting the audio file in the right location plus the default audio location

Remember what we said earlier about consolidating audio and other resources into a single location? Well, we have created the following directory:

```
C:\Cisco\CVP\Tomcat 4.1\webapps\CVP\audio\HelloWorld2
```

We have saved our .wav file as Hello.wav in that directory. Remember that we have also set the default audio path (see Figure 4-106) for our application to /CVP/audio/HelloWorld2. In studio, we can reference our audio files simply by naming them, no need to use a fully qualified path.



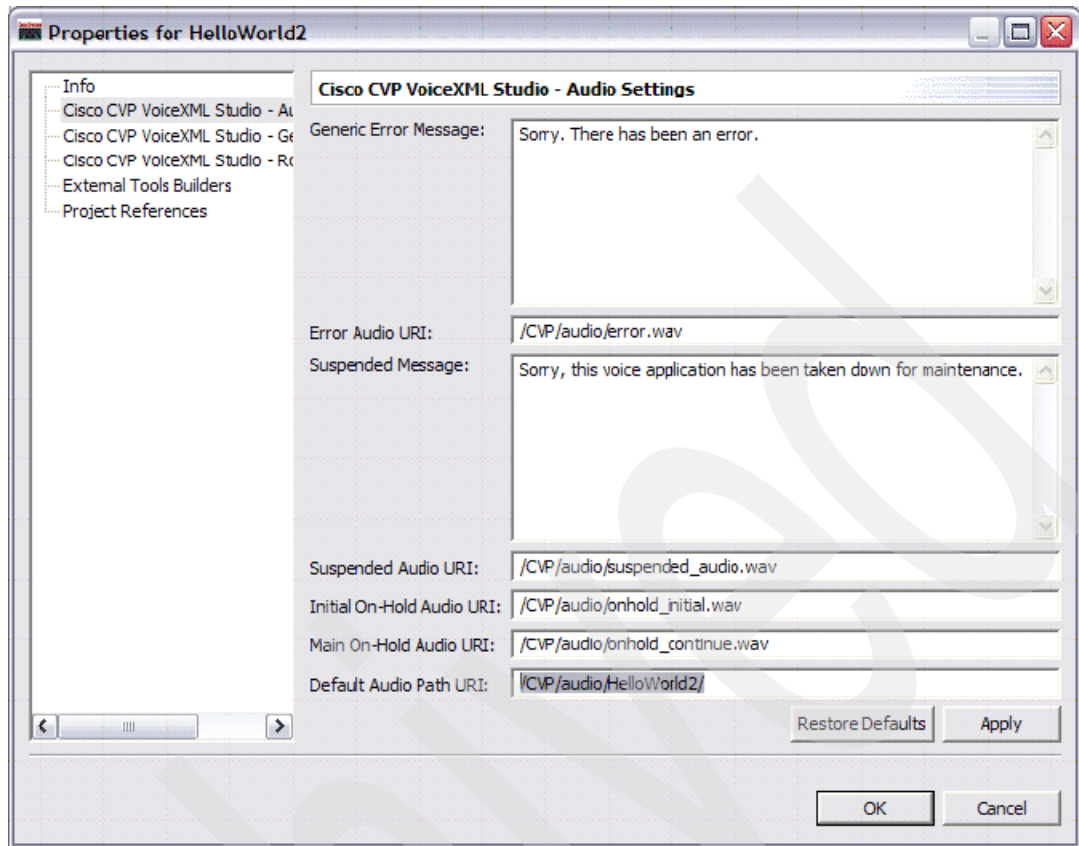


Figure 4-106 Cisco CVP VoiceXML Studio: Audio Settings

All we have to do is duplicate the call flow we had in our HelloWorld1 application . You can simply copy and paste the call flow from HelloWorld1 to the call flow for HelloWorld2. We could simply rebuild the call flow from scratch, but this way permits us to review some handy time-saving concepts as seen in Figure 4-107 on page 192, Figure 4-108 on page 192, Figure 4-109 on page 193, and Figure 4-110 on page 193.

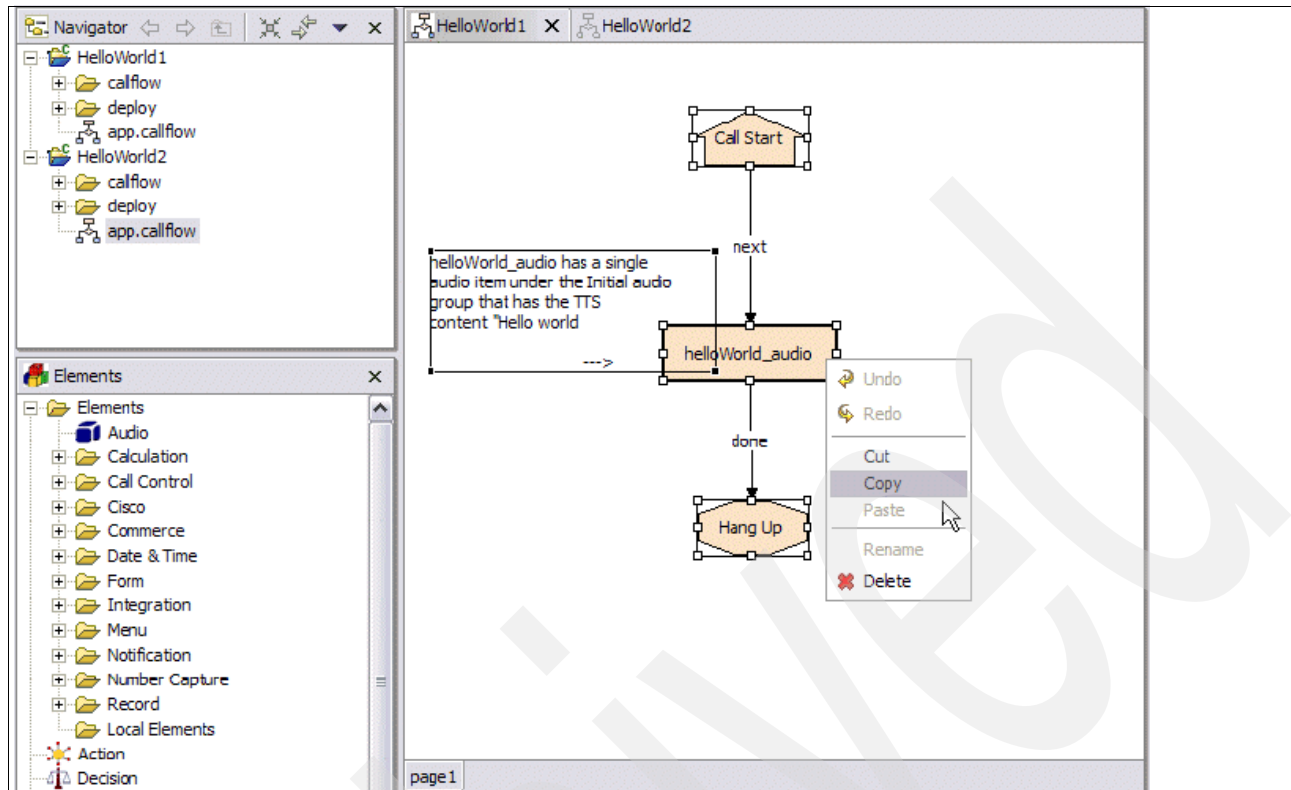


Figure 4-107 HelloWorld1: Select all elements in the call flow for HelloWorld1 then copy them.

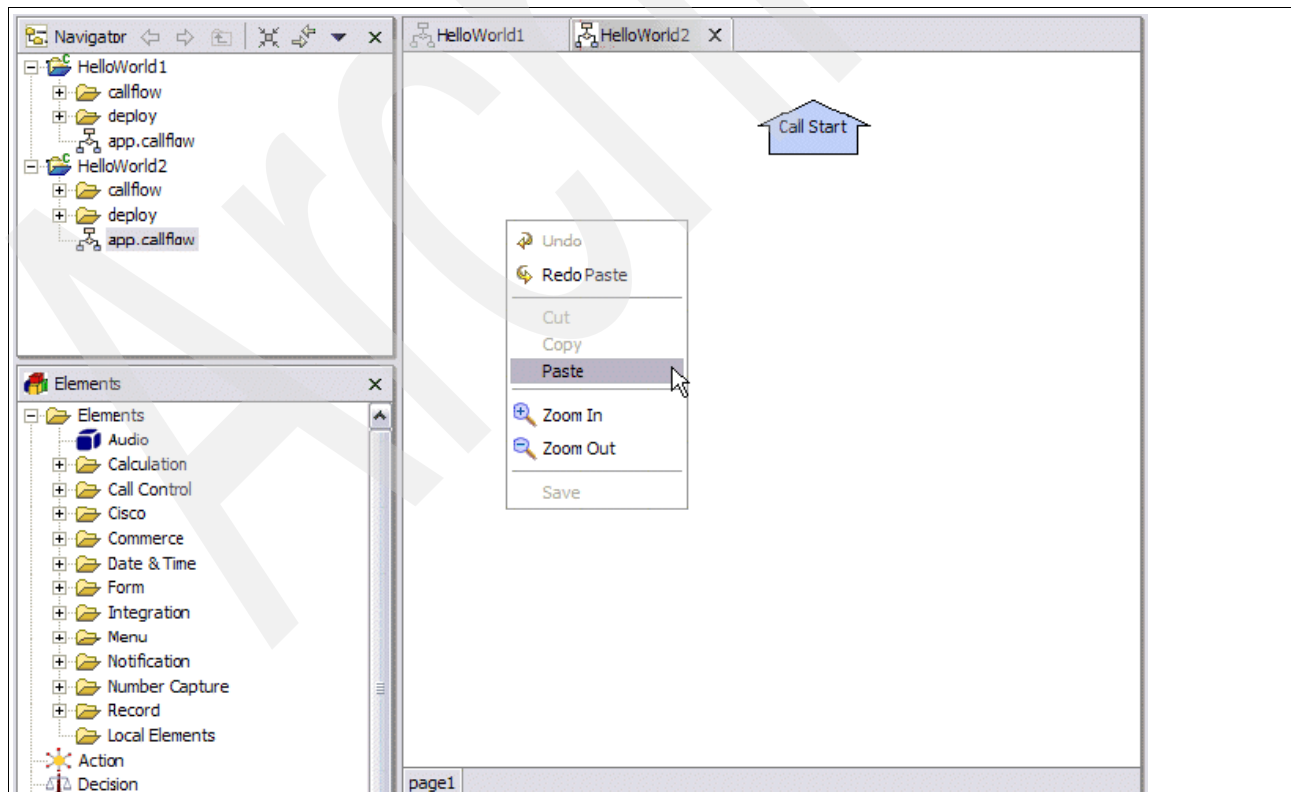


Figure 4-108 HelloWorld2: Paste the copied elements into the call flow for HelloWorld2

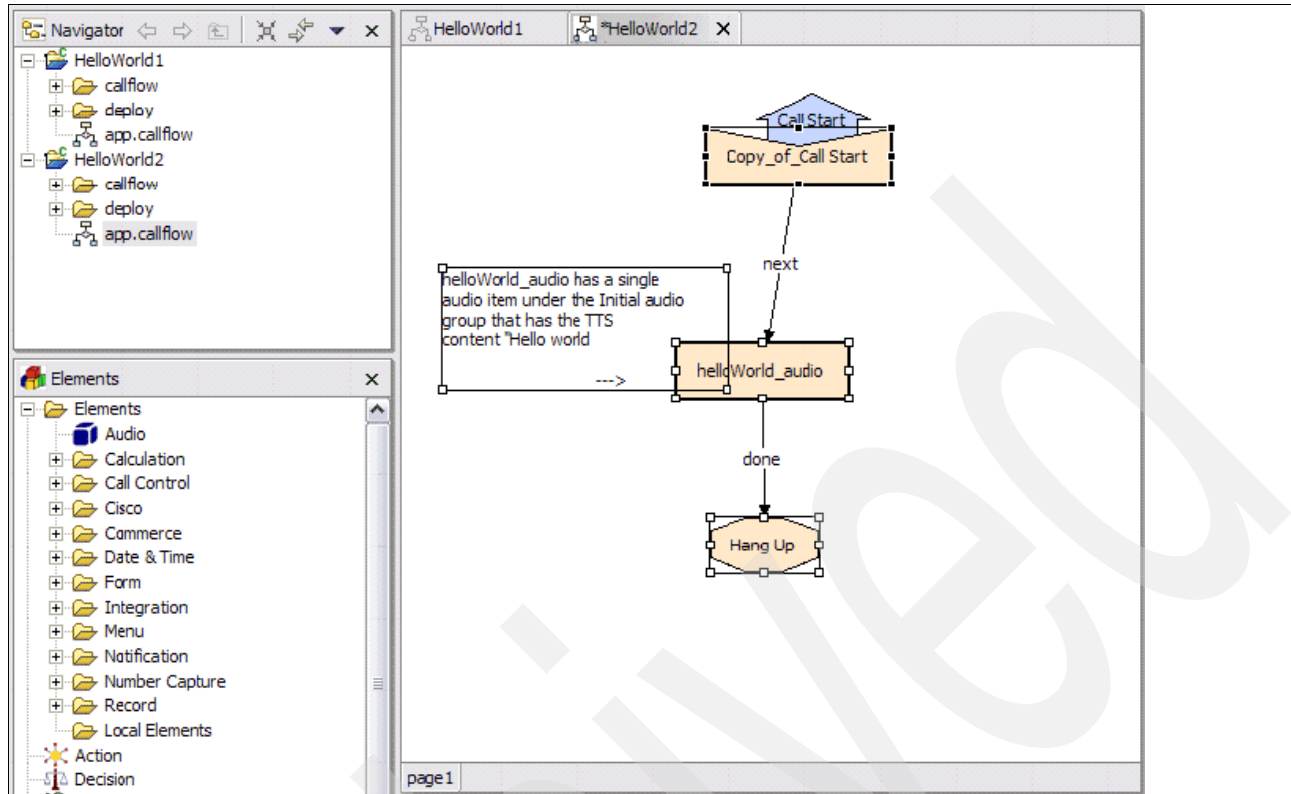


Figure 4-109 HelloWorld2: A copy of the elements from HelloWorld1 now exist in HelloWorld2

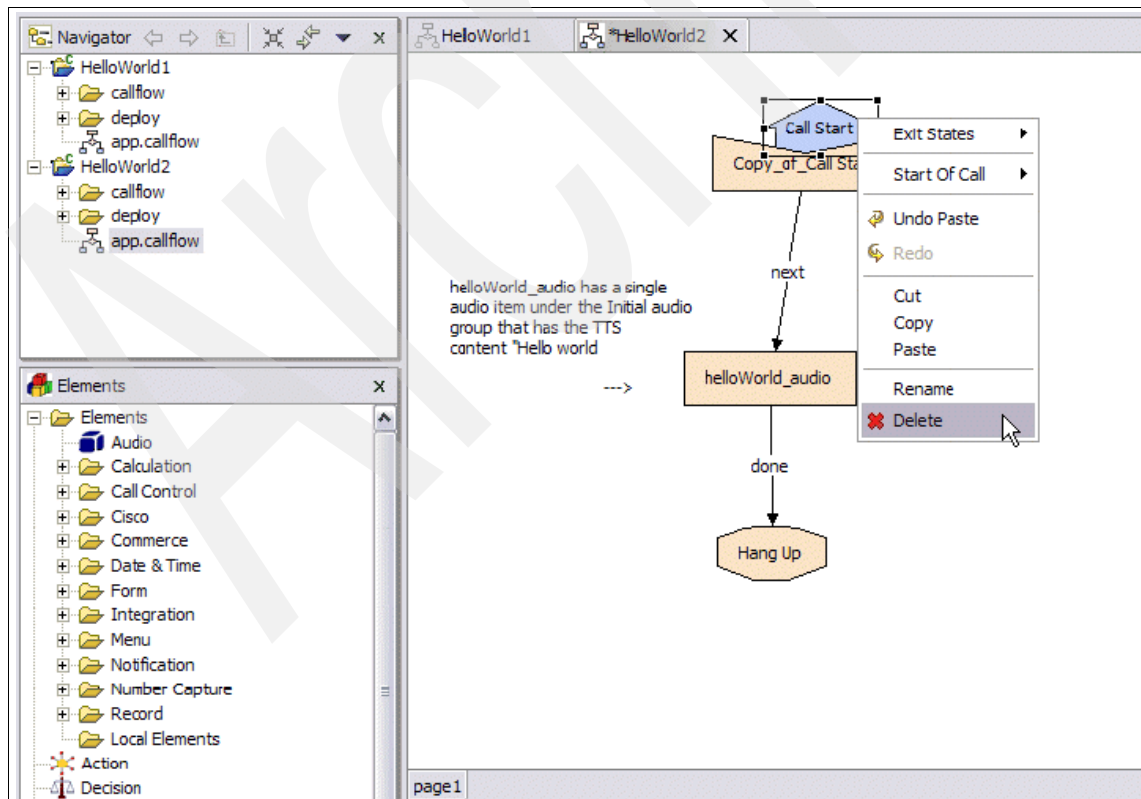


Figure 4-110 HelloWorld2: Deleting the Call Start element

Get rid of the default Call Start element from HelloWorld2. We can work with the one from HelloWorld1, which has reverted to a normal page entry and because each application can only have one Call Start element, we will have to change this as shown in Figure 4-111, Figure 4-112 on page 195, and Figure 4-113 on page 195.

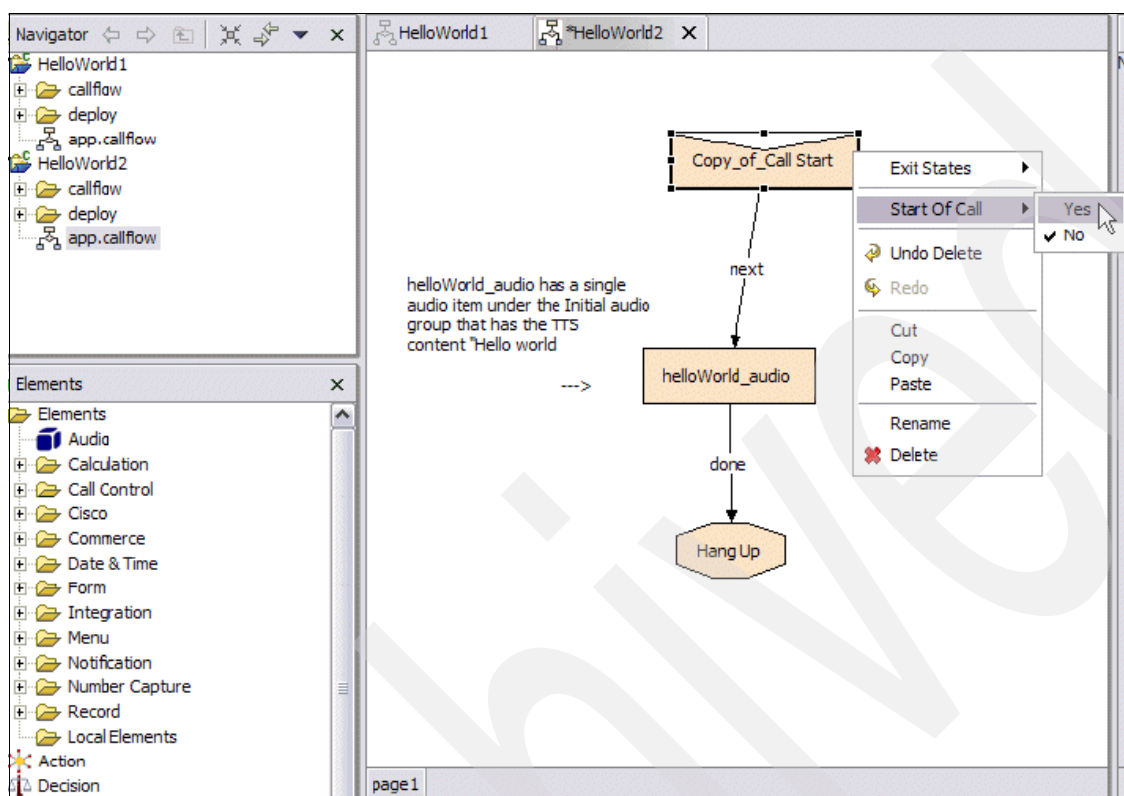


Figure 4-111 HelloWorld2: Changing the Copy\_of\_Call Start element to a “Start Of Call” page entry

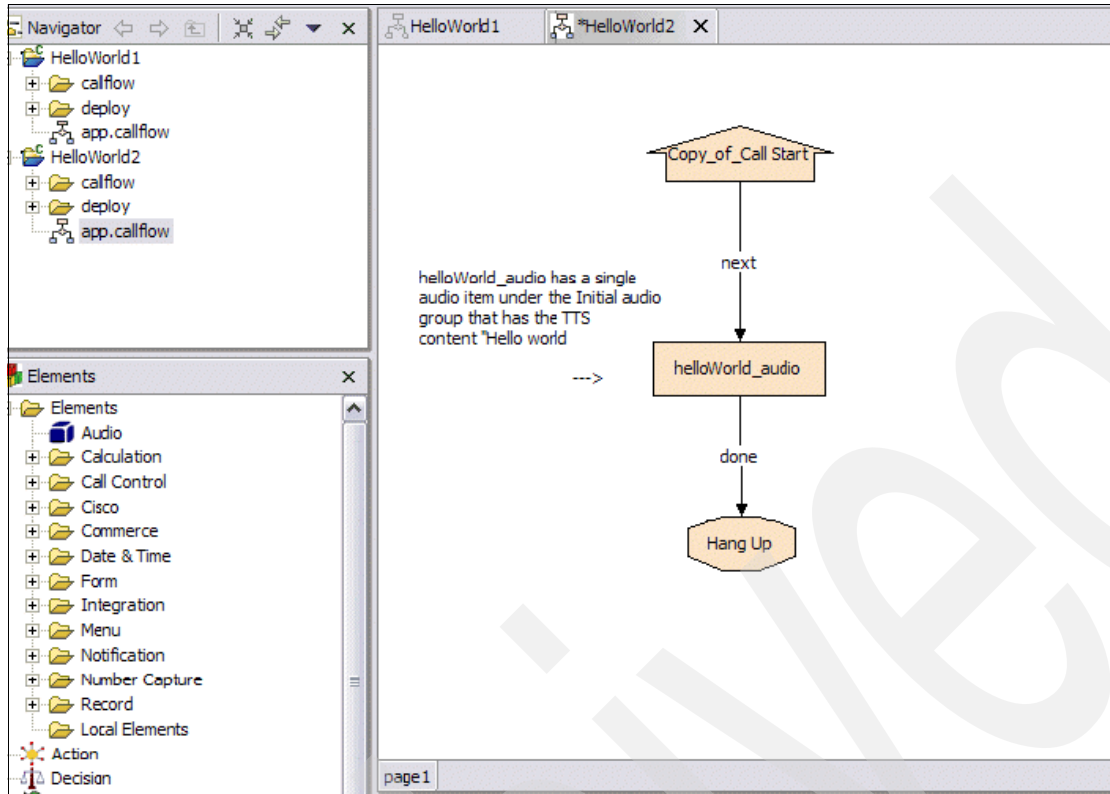


Figure 4-112 HelloWorld2: Result of changing the Copy\_of\_Call Start element to a “Start Of Call” page entry

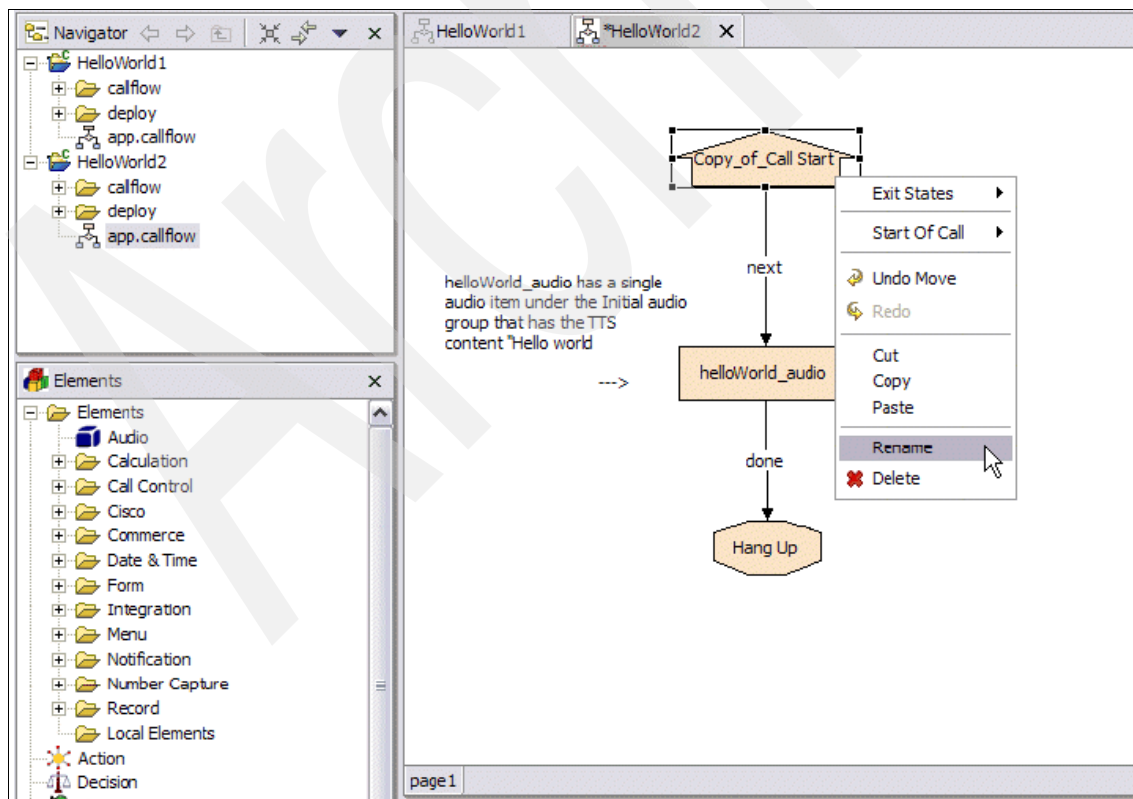


Figure 4-113 HelloWorld2: Renaming the call start element

Modify the comment (see Figure 4-114 and Figure 4-115) so that it is an accurate reflection of what we are doing in HelloWorld2.

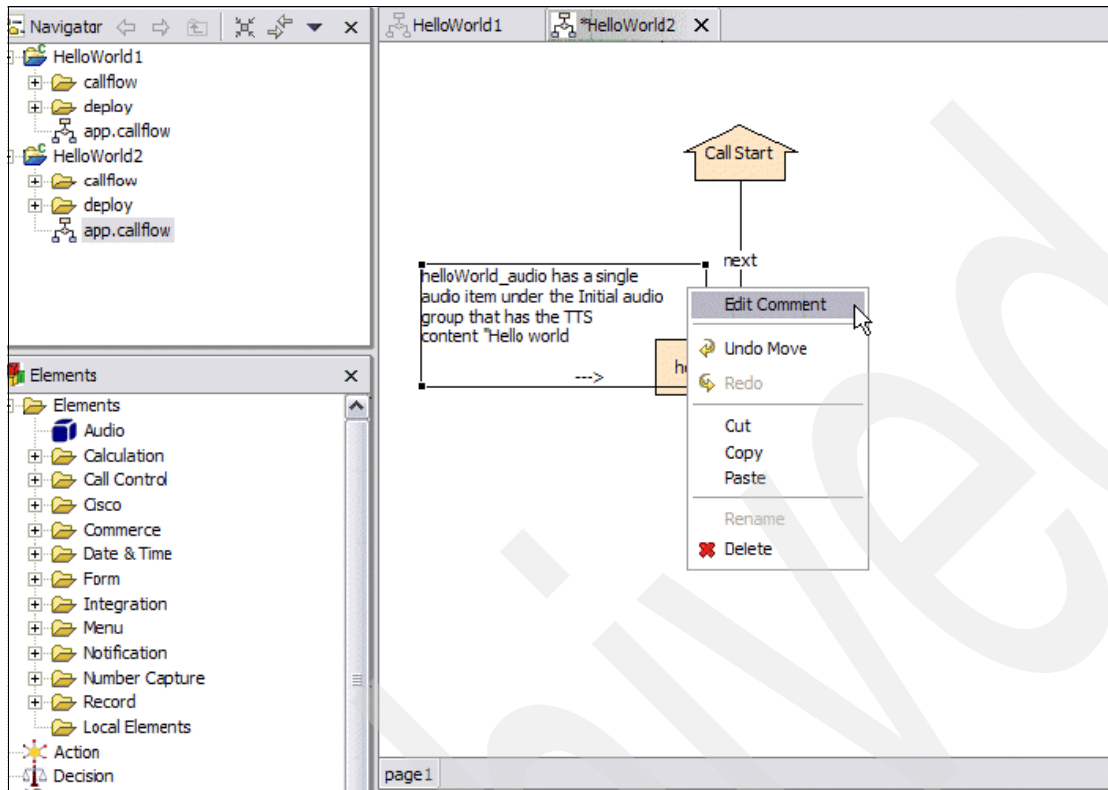


Figure 4-114 HelloWorld2: Editing the Comment

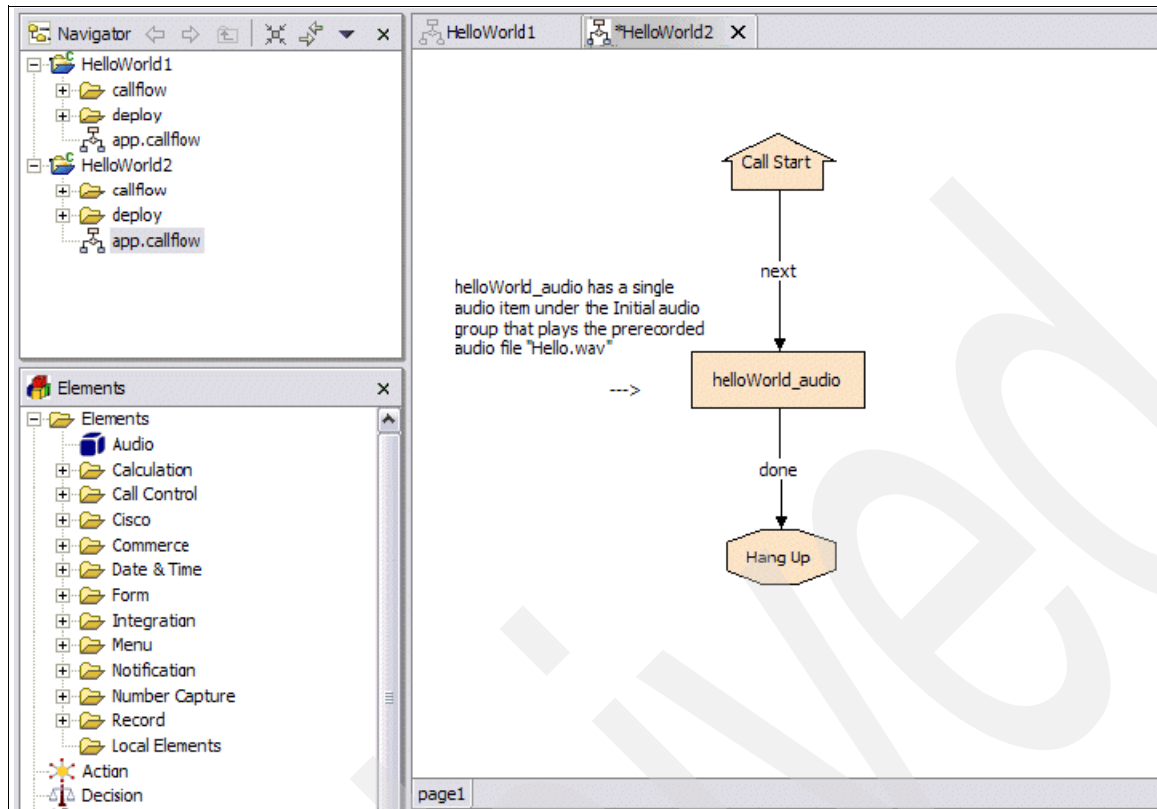


Figure 4-115 HelloWorld2: The edited comment

We now have our skeleton for HelloWorld2. We could also have simply copied the HelloWorld1 project in the navigator pane and called the copy HelloWorld2.

Modify the helloWorld\_audio audio element so that it plays a prerecorded audio file instead of reading back a TTS string as shown in Figure 4-116.



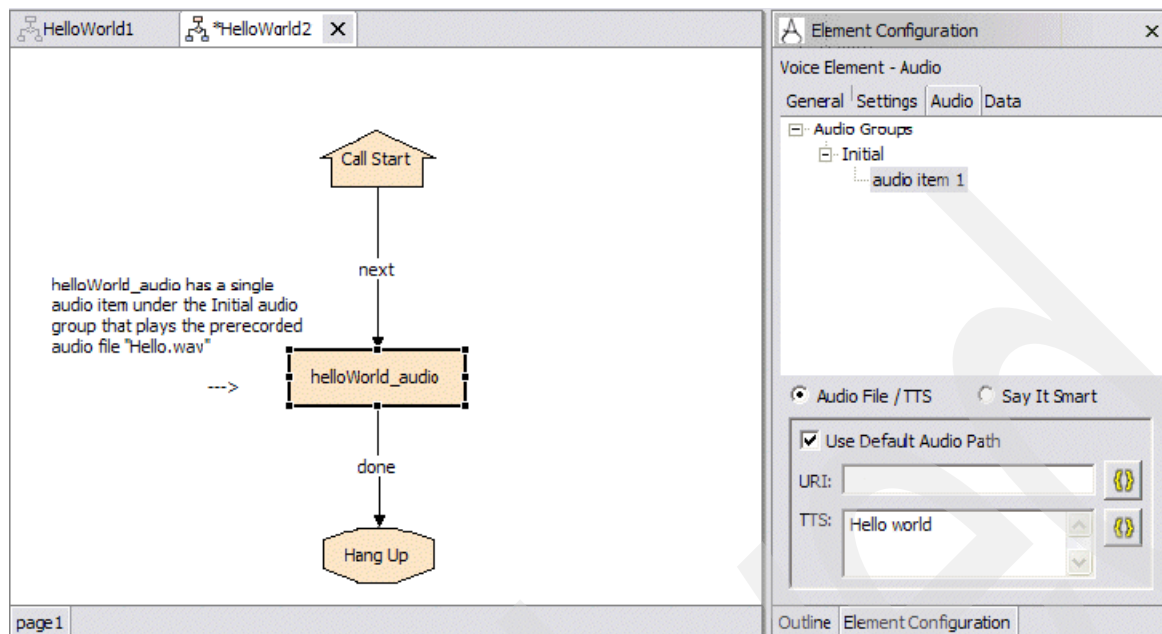


Figure 4-116 HelloWorld2: Modifying the helloWorld\_audio element to play prerecorded audio

With this configuration, our audio element will read back the TTS string Hello World. Change this now as shown in Figure 4-117.

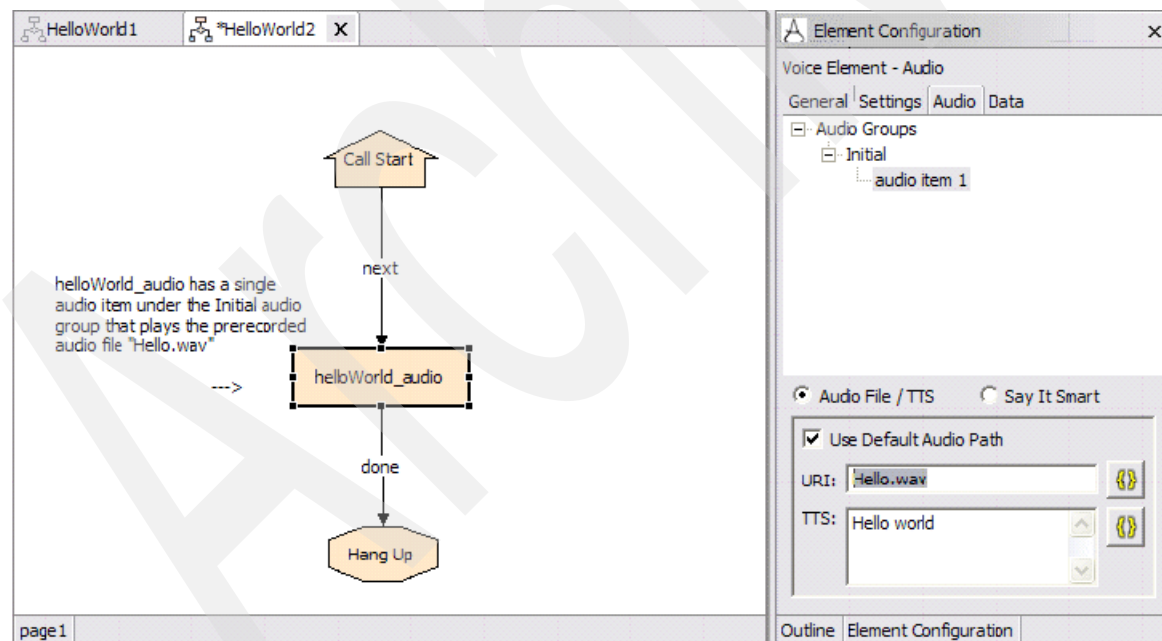


Figure 4-117 HelloWorld2: Setting the URI to point to the prerecorded audio file

Now we have specified an audio file for our element to play. When deployed, the voice browser on the gateway will look in the default audio path (/CVP/audio/HelloWorld2) for the Hello.wav file. If it finds the file and it is an appropriate format, the application will play this prompt. If the browser cannot find Hello.wav, it will fallback to TTS functionality and read the string Hello world. Note that it is often a good practice to provide a TTS backup for static audio with content you know will not change.



You can validate, deploy and test your application. You now have a HelloWorld2 application that plays a prerecorded audio file with a TTS backup.

**Troubleshooting:** You can test that TTS backup is working by temporarily removing the Hello.wav file from your default audio directory. In theory, what should happen is instead of hearing the audio file played back, you should hear the WebSphere Voice Server reading TTS content. But it is very likely that you *will not!* If you do not notice any change and you are sure that you have removed the file, you have encountered an important phenomenon of which to be aware: the gateway will sometimes cache resources such as audio files and grammars for a predefined period so that it does not have to constantly retrieve them from the application server. This can be a source of problems during the development process when you make frequent changes. If you are certain you have changed something and updated your application appropriately, but you do not see or hear any observable changes, you might want to take a look at the gateway and the voice servers caches to see if this is the source of the problem.

On a Cisco gateway, you can view cached audio files by issuing the following command (see Figure 4-118):

```
show http client cache
```

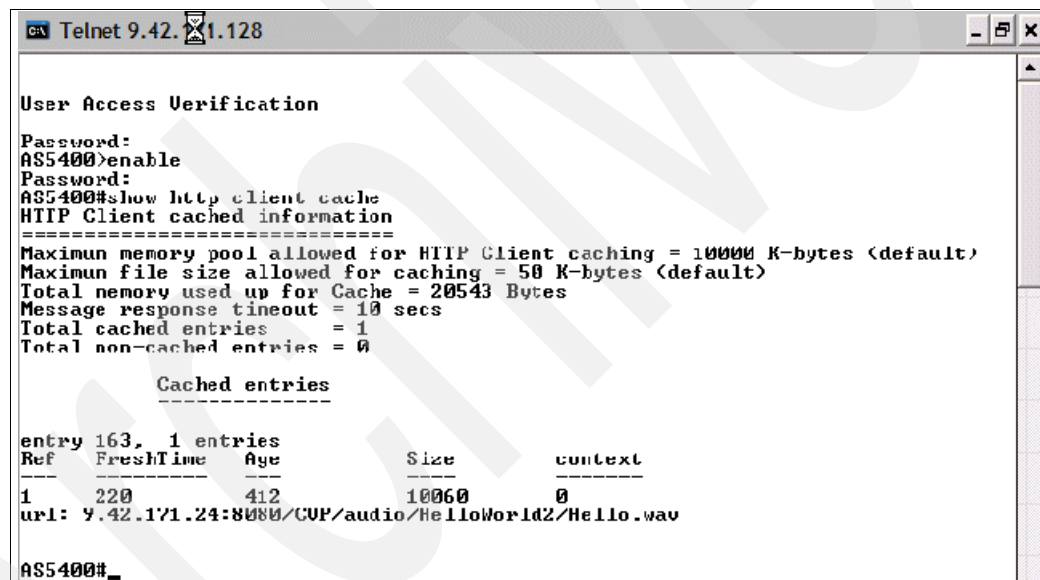


Figure 4-118 Result of show http client cache command

The audio file is no longer present on the Cisco CVP server but it is cached on the gateway which explains why our application is exhibiting unexplained behavior.

Let us flush the HTTP client cache and see what happens now.

```
Router# clear ip http client cache all
```

The HTTP client cache is now clear. Try running HelloWorld2 again and you will see that TTS content is now used as we would expect.

### HelloWorld3: add a form with an inline grammar

To pick up the pace a little bit and start adding some more interesting features to our application, create a new project called HelloWorld3. We are going to make our otherwise static HelloWorld application interactive.

We will have our application greet the user with the phrase Hello world, then ask the user the following question: What is your name please? We will then wait for the user's response. We define an inline grammar for our form element that includes several very common names. If the name the user responds with is recognized by our application, the application responds I recognize that name. If not, the application responds with: I am sorry. I do not recognize that name. If the application is unable to hear the user responses, it will say I am sorry I could not hear your responses. Create the call flow so that it includes a Call Start, a Form, three audio elements and a Hang Up as shown in Figure 4-119. Do not connect any elements just yet.

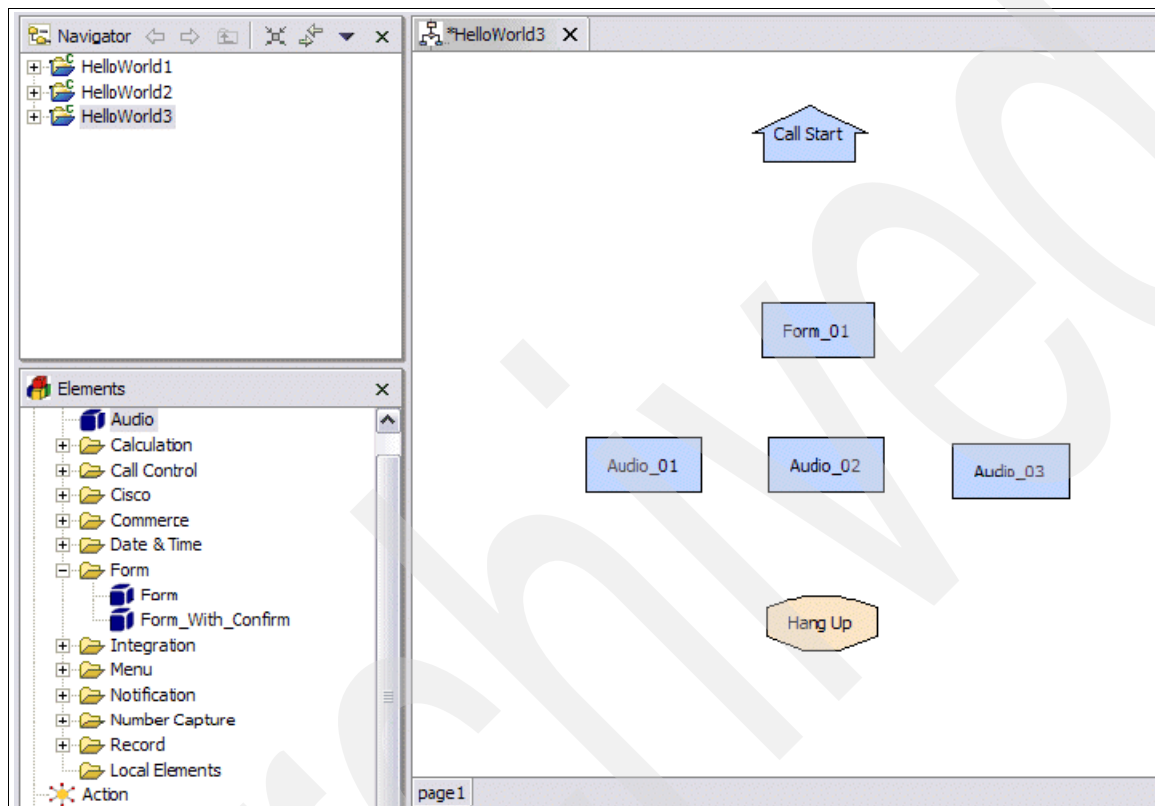


Figure 4-119 HelloWorld3: Creating the call flow for HelloWorld3

1. Give our elements meaningful names as shown in Figure 4-120 on page 201.

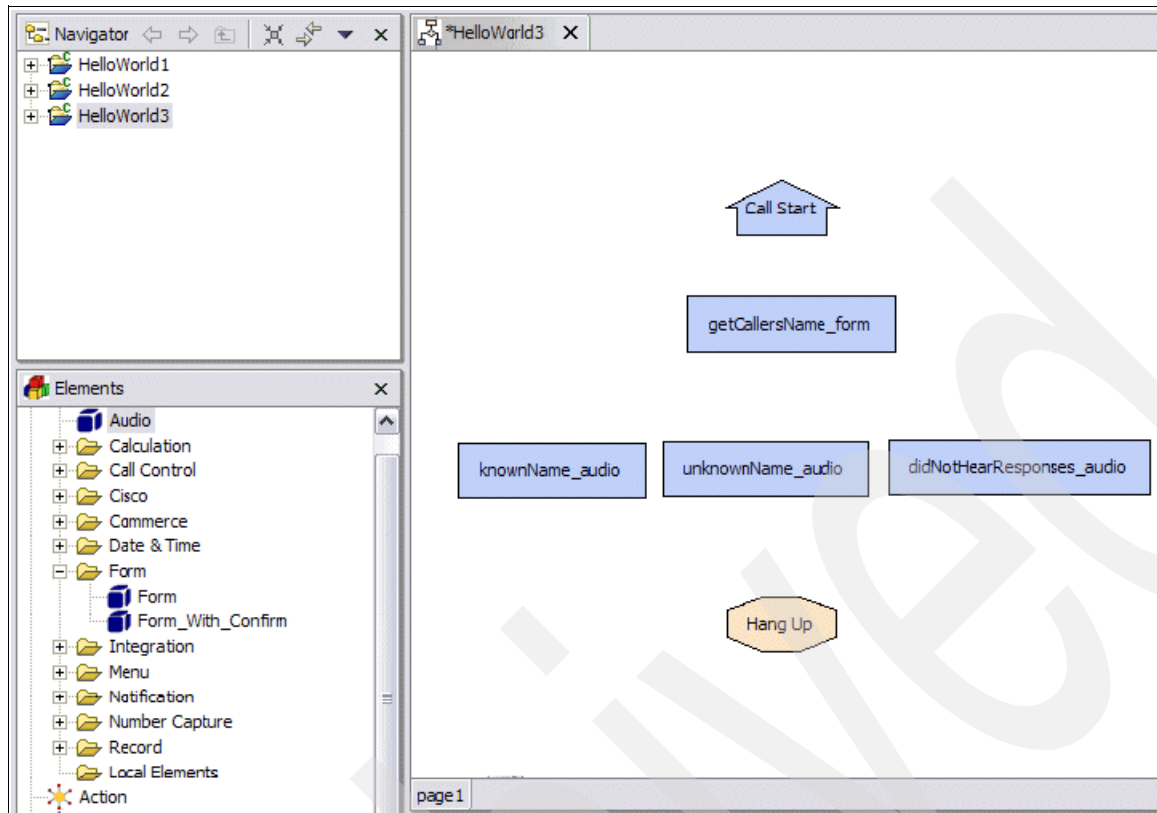


Figure 4-120 HelloWorld3: Naming the elements

2. Start connecting up our exit states in a way that makes sense as shown in Figure 4-121 on page 201. You will see some exit states that you are not familiar with yet. We discuss these shortly.

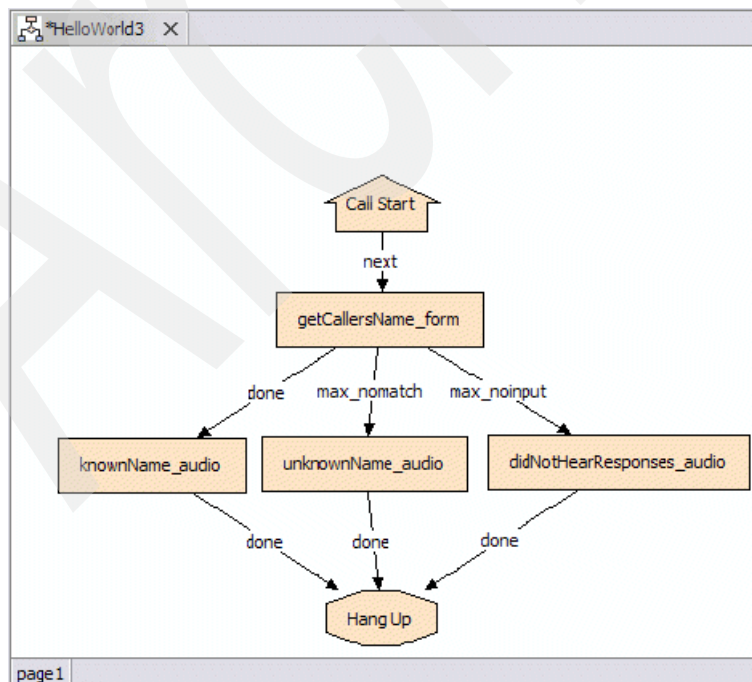


Figure 4-121 HelloWorld3: Connecting exit states

There is nothing really complicated to know about the exit states for our `getCallersName_form`. A *form* is an element that speaks an initial prompt (Hello world. What is your name please?) and then waits for user input. User input is compared against a *grammar*, essentially nothing more than a list of words that our ASR engine can understand within the context of this particular form. If the user utterance is in our grammar and is understood by the form, then the form exits with the exit state `Done`. If the ASR engine cannot hear what the user said for a predefined number of times, the form will have the exit state `max_noInput`. Finally, if the user says a name and the ASR is able to hear the utterance, but does not understand it (not one of the words defined in the form grammar), then the exit state for the form will be `max_noMatch`. So, we have connected our elements such that:

- ▶ If the user utterance is heard and known the `knownName_audio` audio element will play the TTS text: I recognize that name.
  - ▶ If the user utterance is heard but not understood by the ASR engine three times, the `unknownName_audio` element will play the TTS text I am sorry I do not recognize that name.
  - ▶ If the ASR engine cannot hear the user utterance three times in a row, then the `didNotHearResponses_audio` item will say "I am sorry I could not hear your responses."
3. With that in mind, configure our audio items as shown in Figure 4-122 on page 202, Figure 4-123 on page 203, and Figure 4-124 on page 203.

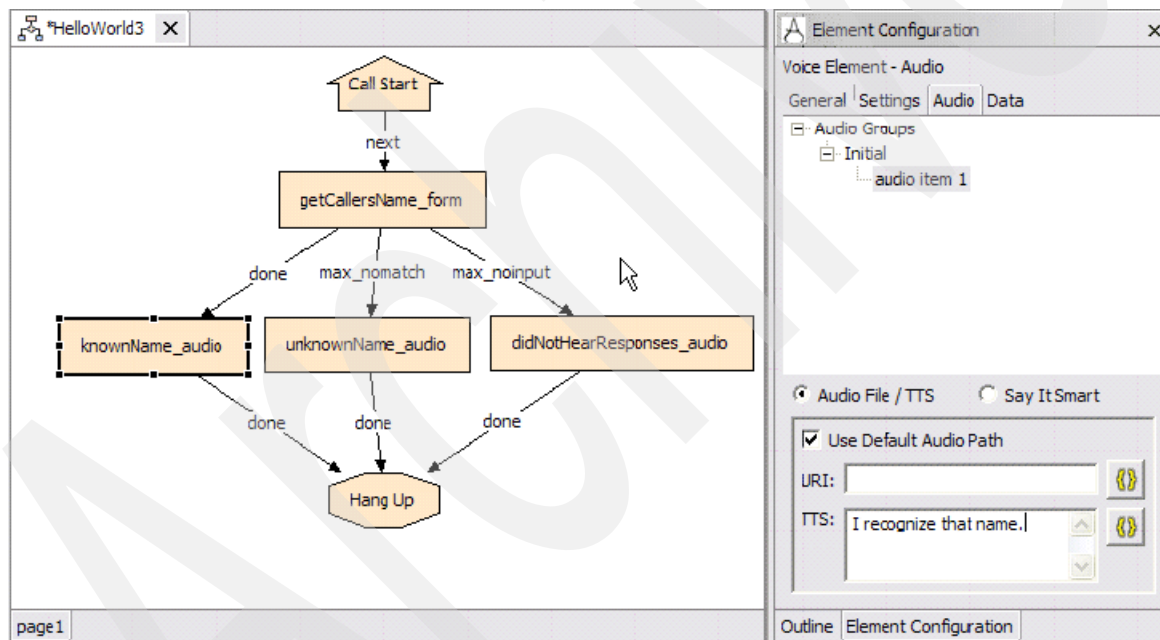


Figure 4-122 HelloWorld3: Configuring the `knownName_audio` element

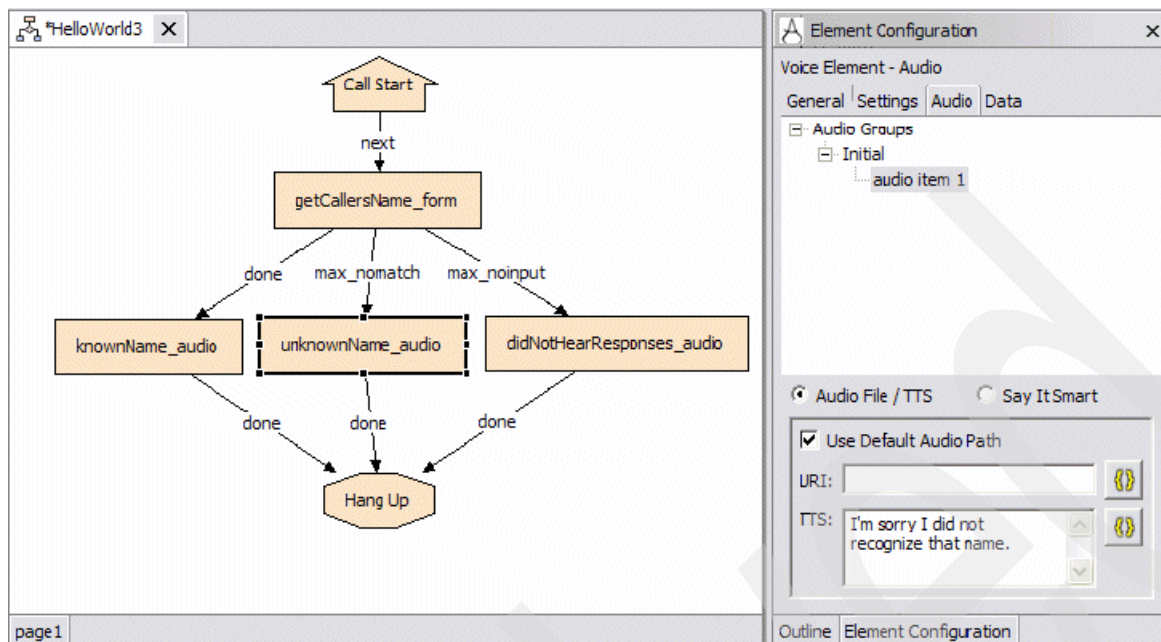


Figure 4-123 HelloWorld3: Configuring the unknownName\_audio element

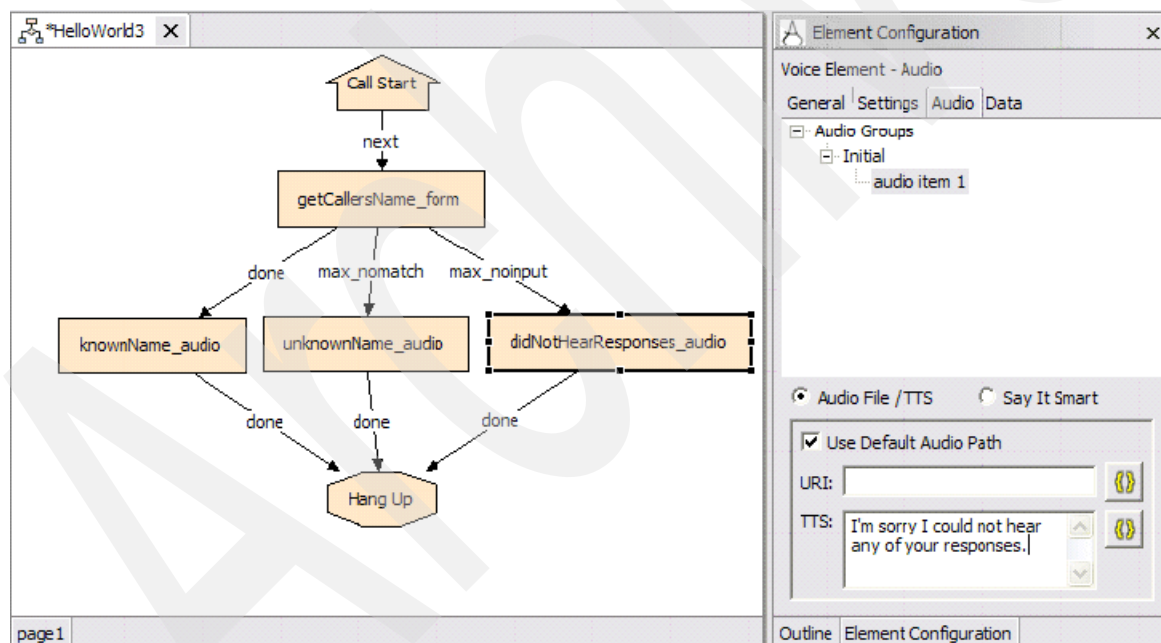


Figure 4-124 HelloWorld3: Configuring the didNotHearResponse\_audio element

4. Our audio items are all set. Configure our form so it exhibits the desired behavior. The first thing we want to do is define the audio items that will be associated with our form. Set up the initial prompt first as shown in Figure 4-125.



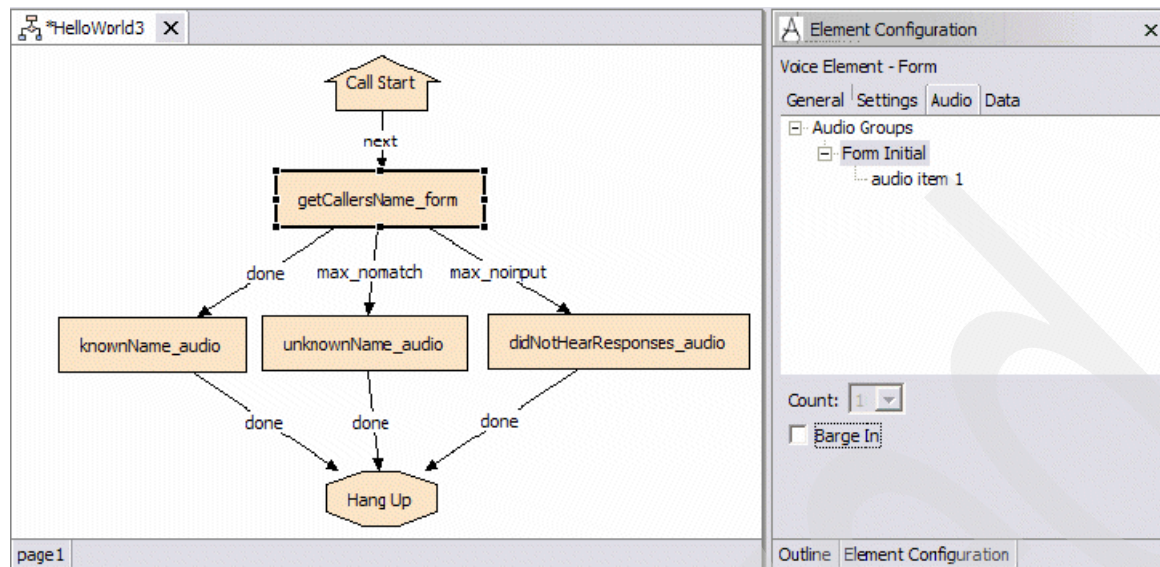


Figure 4-125 HelloWorld3: Disabling barge-in for the getCallerName\_form audio group

5. Disable barge-ins for this audio group. By default ,barge-in is enabled on all audio groups. This allows the user to interrupt the playing of a TTS prompt and proceed to the next step in execution. We find that this can sometimes lead to unexpected results while developing applications, so, as a general rule, we disable barge-in until it is really needed. Preproduction deployment testing comes to mind as a good time to do so. To disable barge-in simply deselect the **Barge In** check box for the audio group in question.

We want our form to say Hello World. What is your name when first visited by a caller (see Figure 4-126 on page 204, Figure 4-127 on page 205, and Figure 4-128 on page 205).

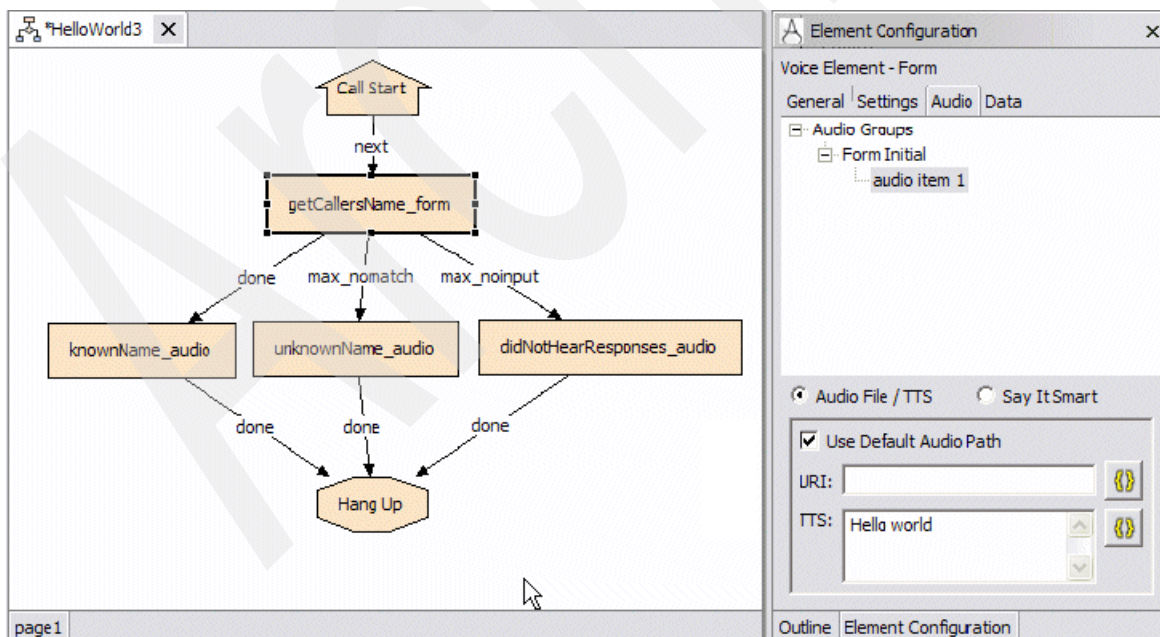


Figure 4-126 HelloWorld3: Defining TTS for audio item 1

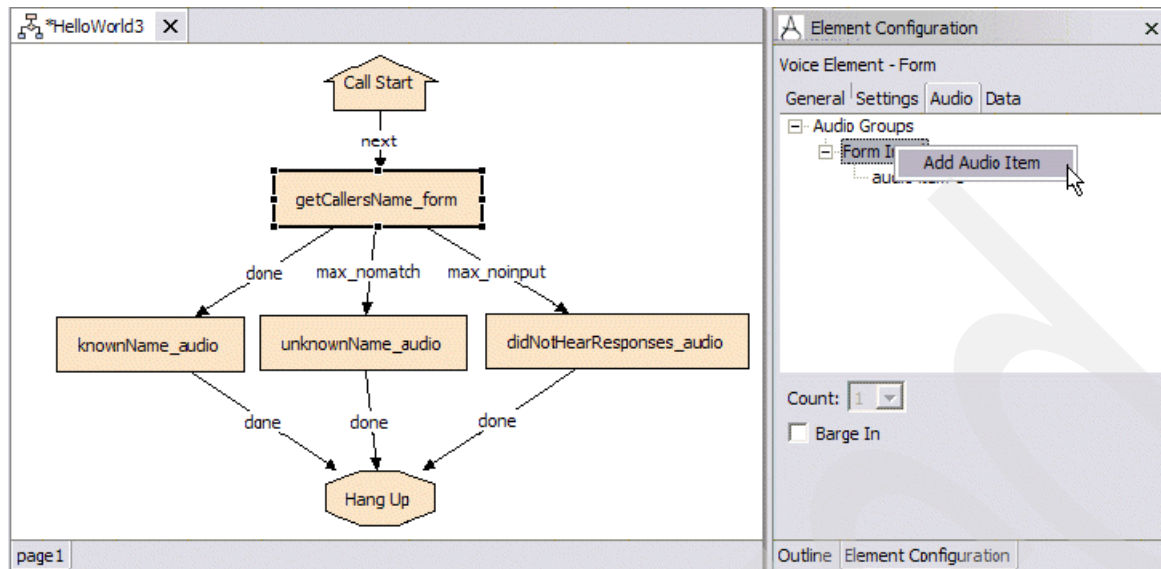


Figure 4-127 HelloWorld3: Adding an audio item

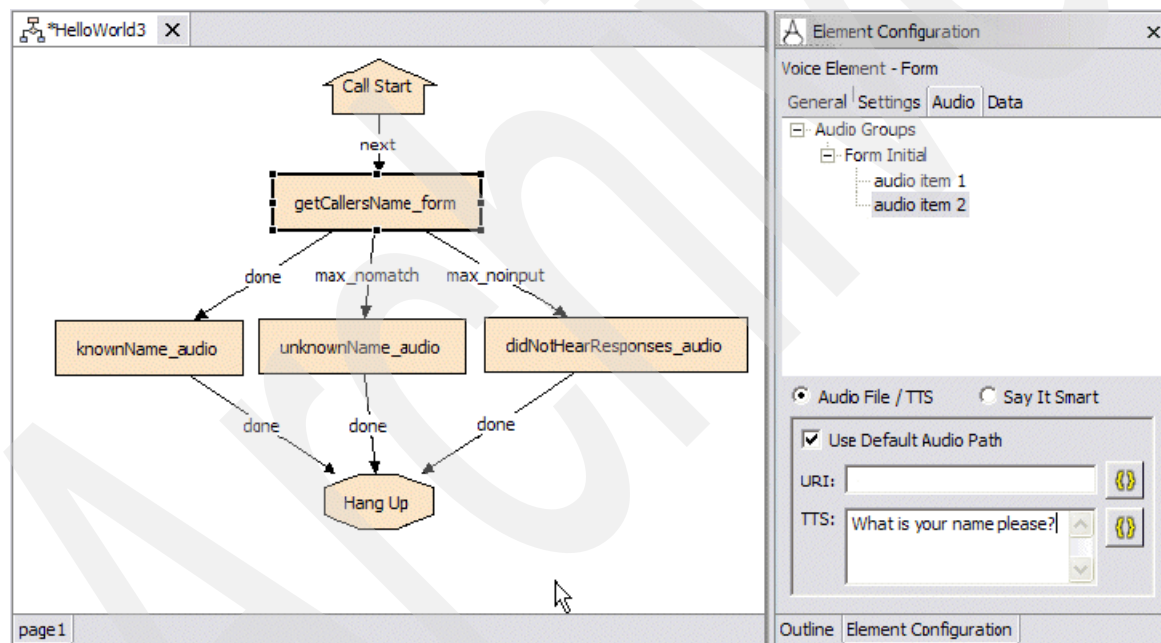


Figure 4-128 HelloWorld3: Defining TTS for audio item 2

Why did not we just put the entire phrase Hello world. What is your name please? into audio item 1, as we normally would have? Well, we certainly could have done that and it would have been far more efficient to do so. However, we did it in the fashion illustrated here to show you that audio groups can consist of multiple audio subitems (as many as the developer pleases). This fact comes in very handy in more advanced applications where the developer has to read back more complex information. We elaborate on this later. For now, rest assured that this configuration simply reads the phrase Hello world. What is your name please back to the caller when the form element is first visited.

We must define some new audio groups that we have not seen yet so that our form will behave in a user friendly manner, as shown in Figure 4-129 on page 206 and Figure 4-130 on page 206.

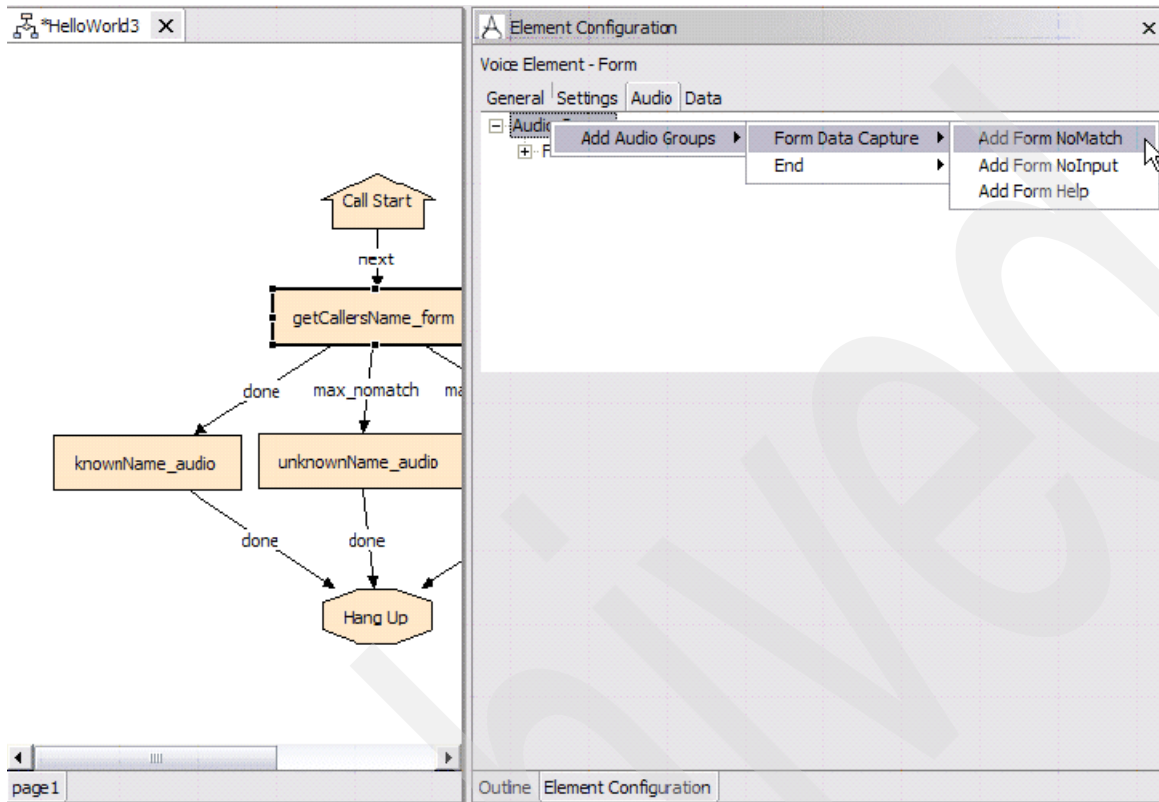


Figure 4-129 HelloWorld3: Adding a Form NoMatch audio group

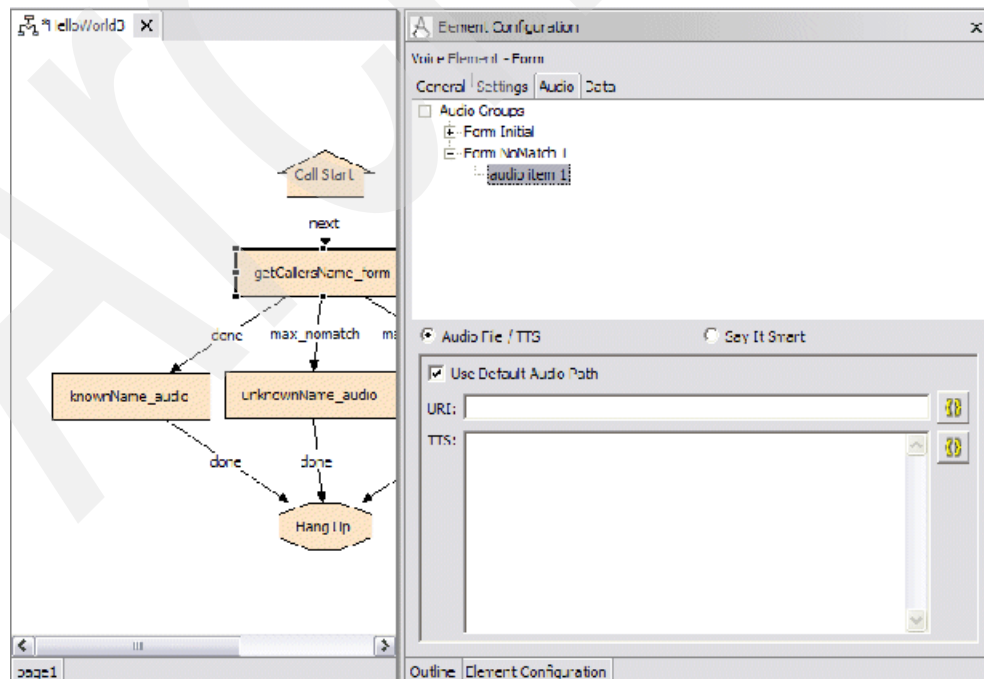


Figure 4-130 HelloWorld3: Result of adding a Form NoMatch audio group



As you can see, the form element only requires that we specify an initial audio group, but there are a number of optional audio groups that can be added to a form element that enhance the functionality and user friendliness of the form. Table 4-5 shows the different audio groups that you can add to a form, what they mean, and how we will set them up for our `getCallerName_form`.

Table 4-5 Audio Groups that can be added to a form

Audio Group Name	Found Under	Purpose	Our TTS Text
Form NoMatch	Form Data Capture	This audio group will be played each time the form encounters a noMatch event. We can add several of these so that a different prompt of increasing severity is played for each consecutive event.	noMatch1: "I am sorry I did not understand what you just said. Please try again"  noMatch2: "I am sorry I did not understand what you just said. Please try again, speak clearly please"  noMatch3: "I am sorry that was the third time I did not understand what you said"
Form NoInput	Form Data Capture	This audio group will be played if no input is detected by the form. We can add several of these as well.	noInput1: "I am sorry I did not hear you say anything"  noInput2: "I am sorry I did not hear you say anything for the second time"  noInput1: "I am sorry I did not hear you say anything for the third time."
Form Help	Form Data Capture	This audio group will be played when the user utters the specified help keyword. This prompt should include instructions on how to respond to the question posed by the form.	"Please say the first name of a person. You can say: Scott, Mary, John, Peter or Steven. If I recognize this name I will say so."
Add Done	End	This audio group will be played when the data capture process is completed successfully (i.e. before exiting with the "Done" exit state)	"Thank you"

6. Add all the audio groups we need for our form by right-clicking the audio groups root node and selecting the appropriate entries. Then, enter the appropriate TTS text for each of

these audio groups. When you are finished, your forms audio configuration should look similar to Figure 4-131 and Figure 4-132. Remember to disable barge-in on all of the audio groups.

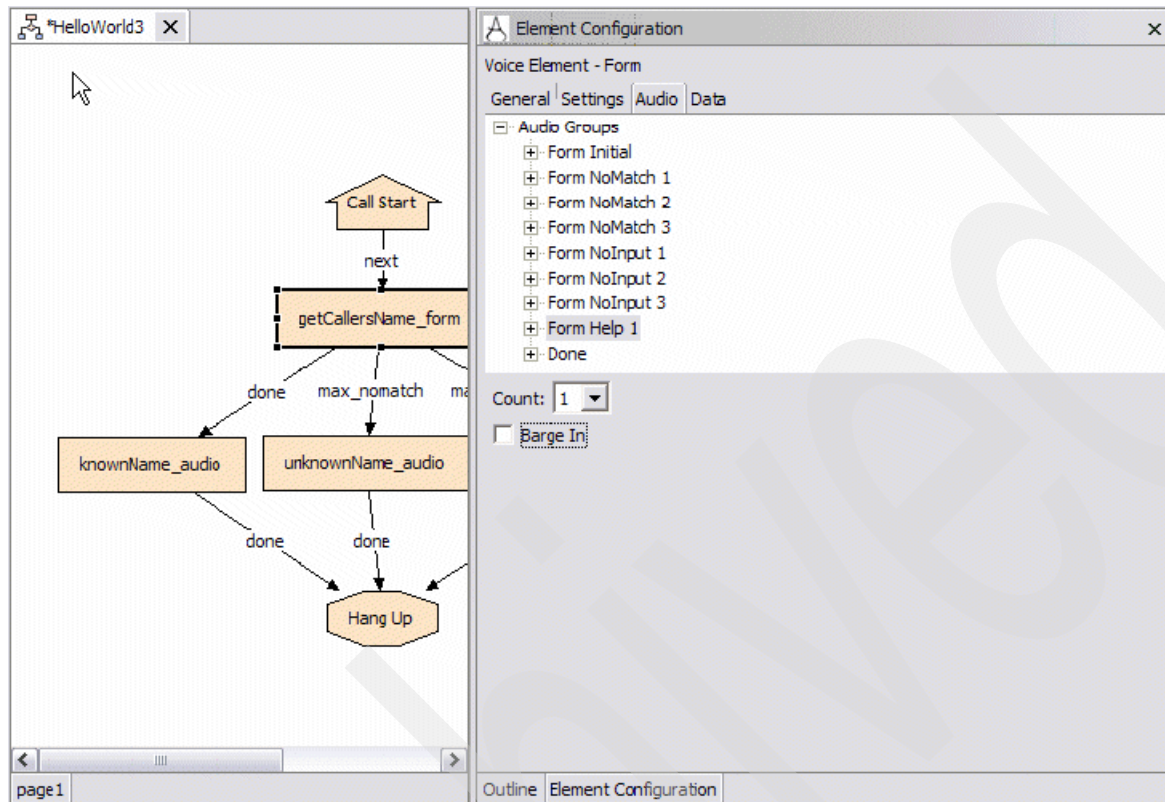


Figure 4-131 HelloWorld3: Form Help 1 audio group

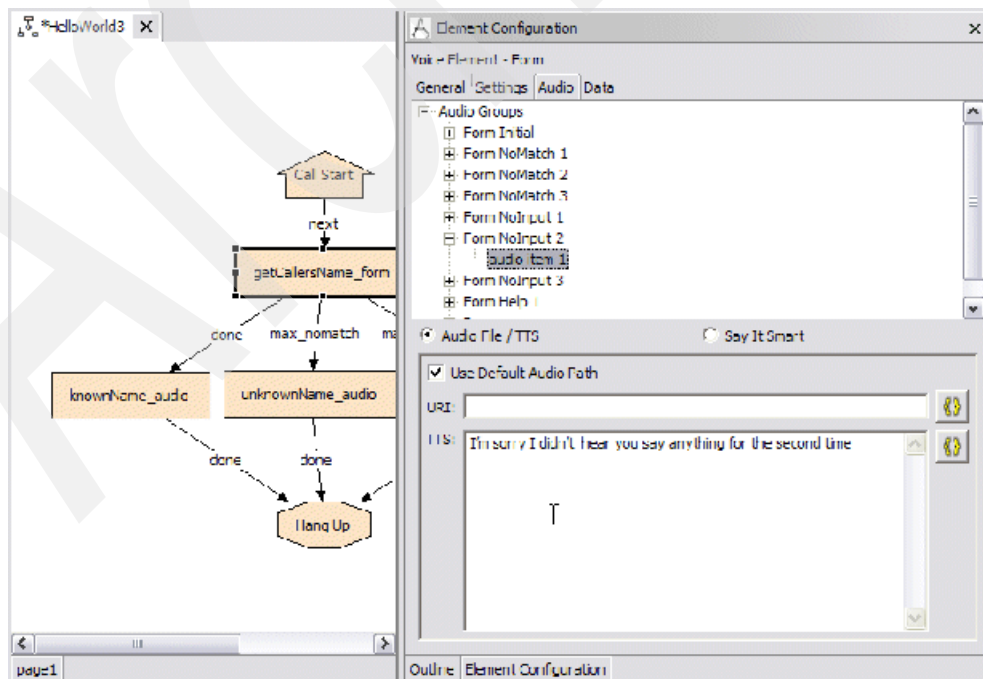


Figure 4-132 HelloWorld3: Form NoInput 2 audio group - audio item 1

Now, we have set up our audio successfully. Go ahead and start looking at other important settings.

7. First, specify what type of input this form is able to receive (see Figure 4-133 and Figure 4-134 on page 210). The possible values are: voice, dtmf, or both. DTMF input is clearly not appropriate for a form asking for a name. So, **voice** only.

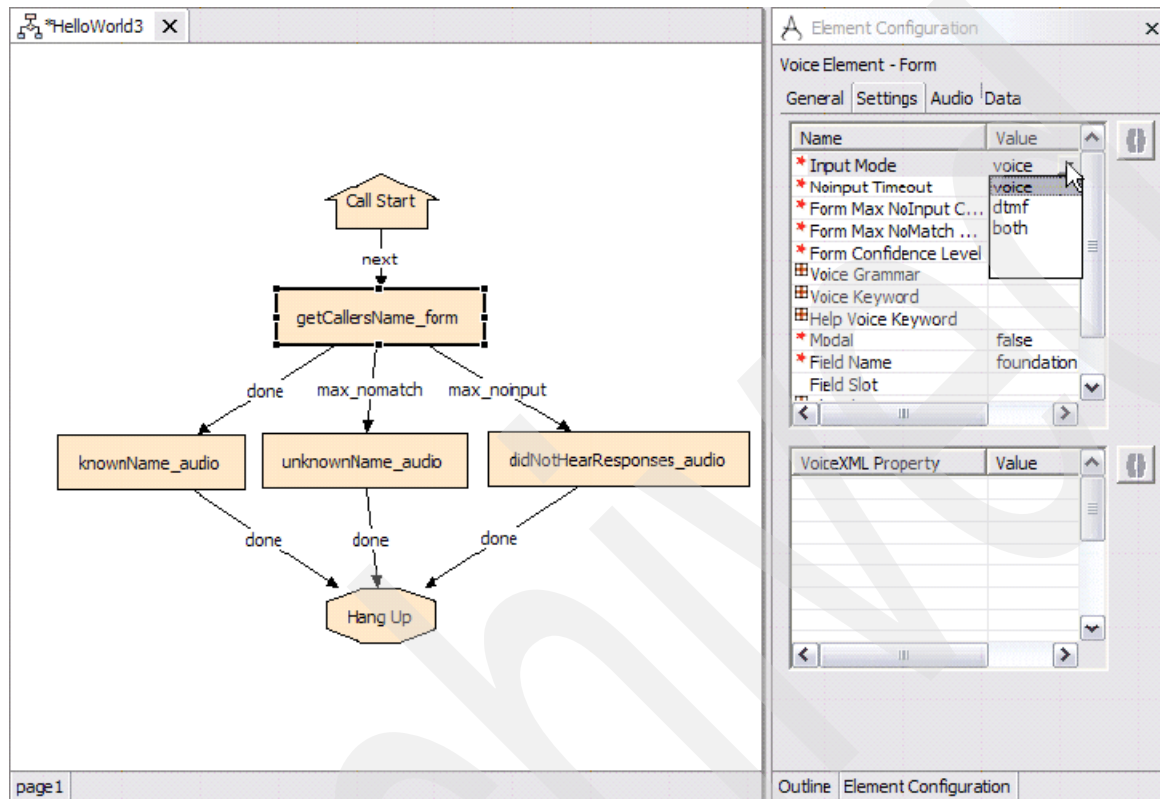


Figure 4-133 HelloWorld3: Setting the voice only attribute

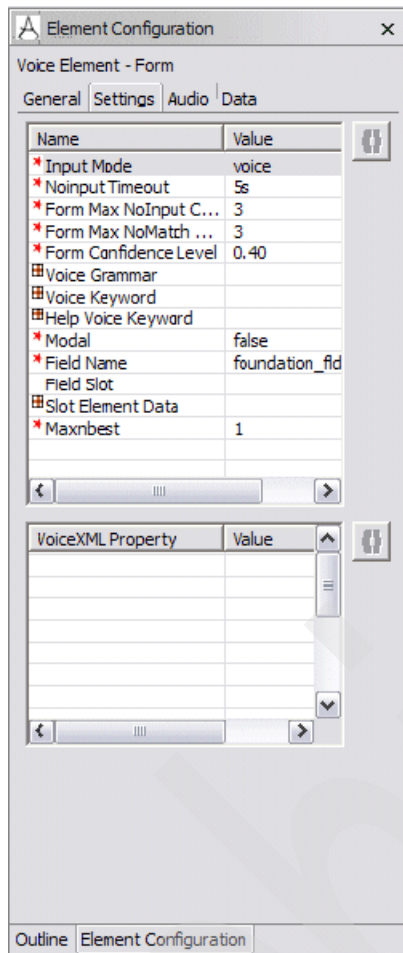


Figure 4-134 Voice Element - Form Settings

There are a number of other setting we can play with for a form. This is true for every configurable element. If you want a comprehensive listing of all settings associated with a particular element and what those settings mean, you should consult the “*Cisco CVP VoiceXML V3.1 Element Specifications*”. We mention some of the more important form settings as shown in Table 4-6.

Table 4-6 Voice Element - Form Settings

Setting Name	Meaning	Value for our application
Input Mode	The type of entry allowed for input. Possible values are: voice   dtmf   both.	voice
Noinput Timeout	The maximum time length allowed (in seconds) for silence or no dtmf entry before a noinput event is thrown.	Leave it at the default value of 5 seconds
Form Max NoInput	0 = infinite noinputs allowed.	We will allow for 3 no input events before we give up
Form Max NoMatch	0 = infinite nomatch allowed.	We will allow for 3 no match events before we give up

Setting Name	Meaning	Value for our application
Form Confidence Level	The confidence level threshold to use for data capture. This value will be returned by the ASR engine for every match it makes.	Leave this at the default value of 0.40, this is a nice safe value that will recognize most utterances without the danger of returning many false positives. However, you might have to tweak and fine-tune this value to obtain satisfactorily precise results.

All that remains to do is for us to define an inline grammar for our form. This is a really nice feature of studio. For simple forms, all we need do is point and click to define keywords that the form will place in an inline grammar.

- Define the names Scott, Mary, John, Peter, and Steven as names our form will recognize. Just add Voice Keywords by right-clicking and choosing **Add Voice Keyword** as shown in Figure 4-135, Figure 4-136 on page 212, and Figure 4-137 on page 213.

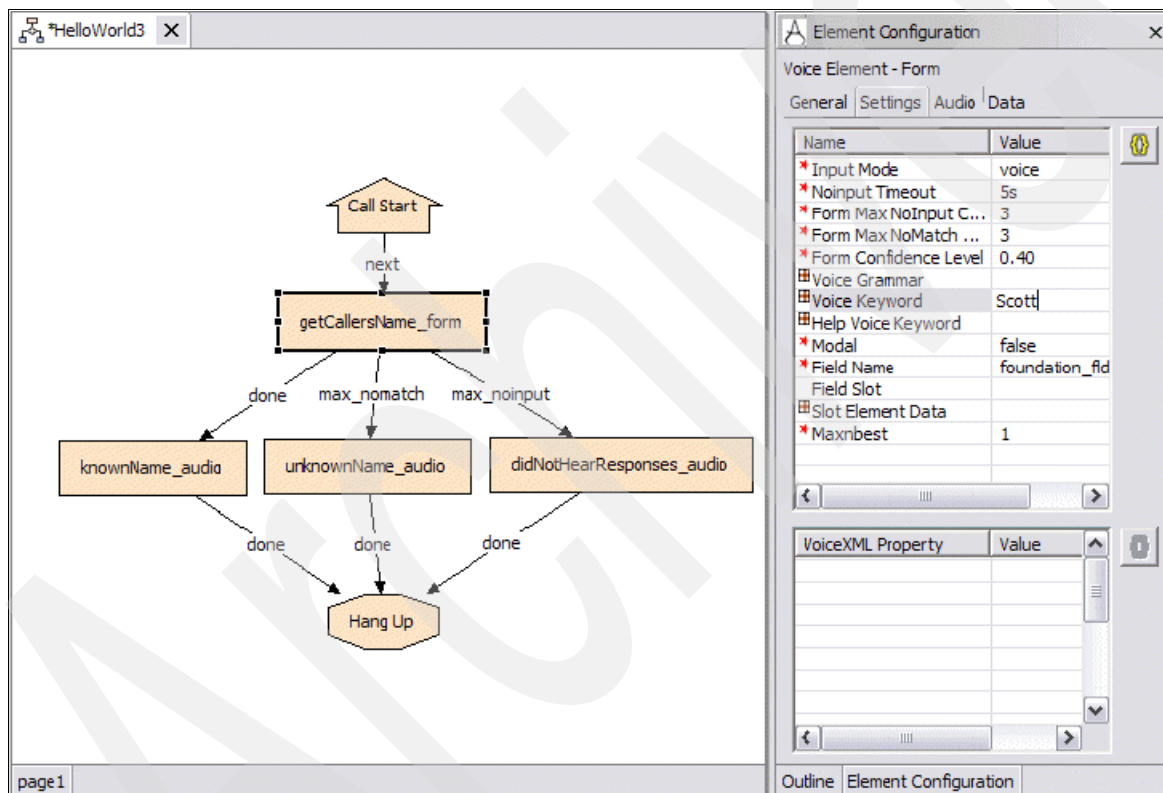


Figure 4-135 HelloWorld3: Setting the Voice Keyword to "Scott"

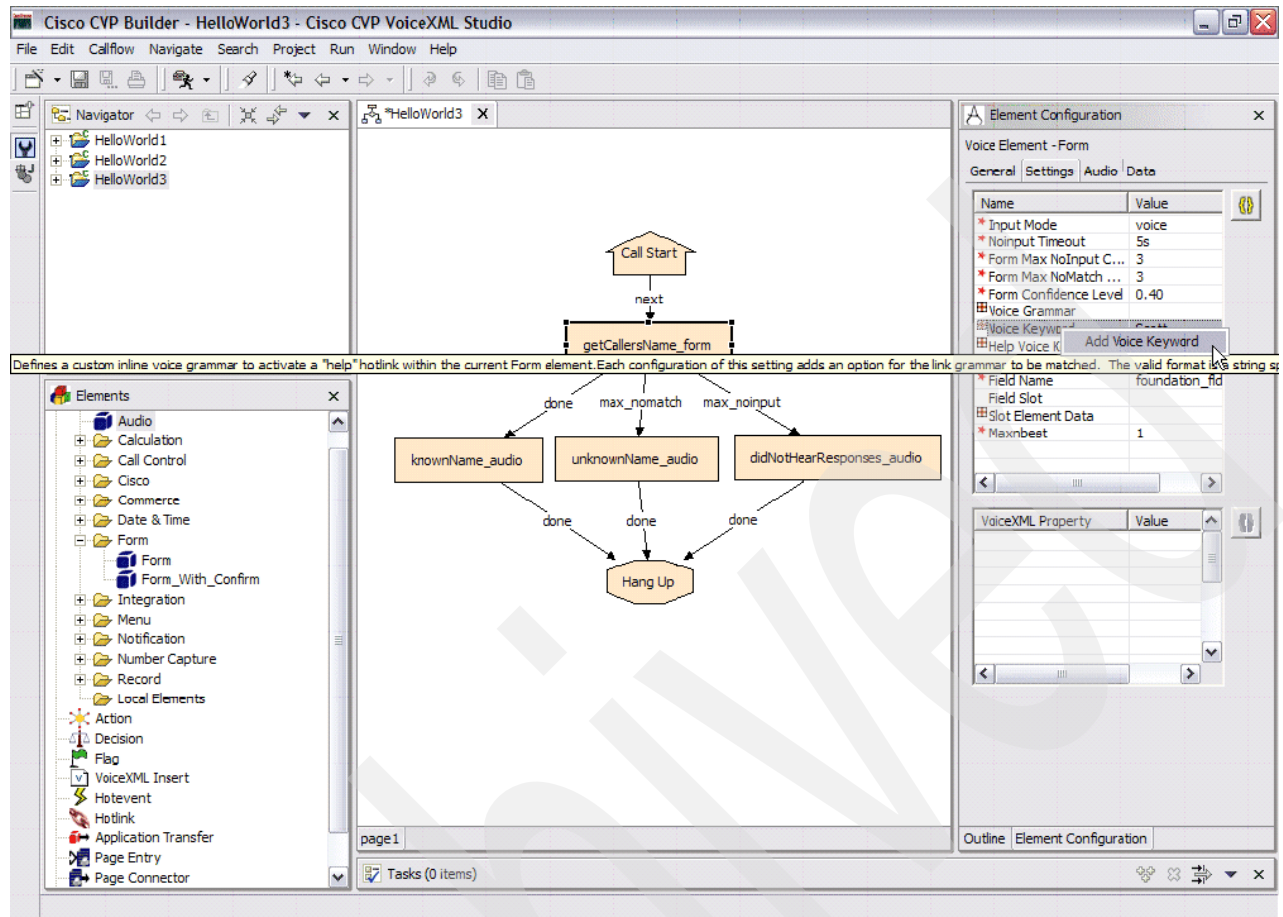


Figure 4-136 HelloWorld3: Adding a second voice keyword



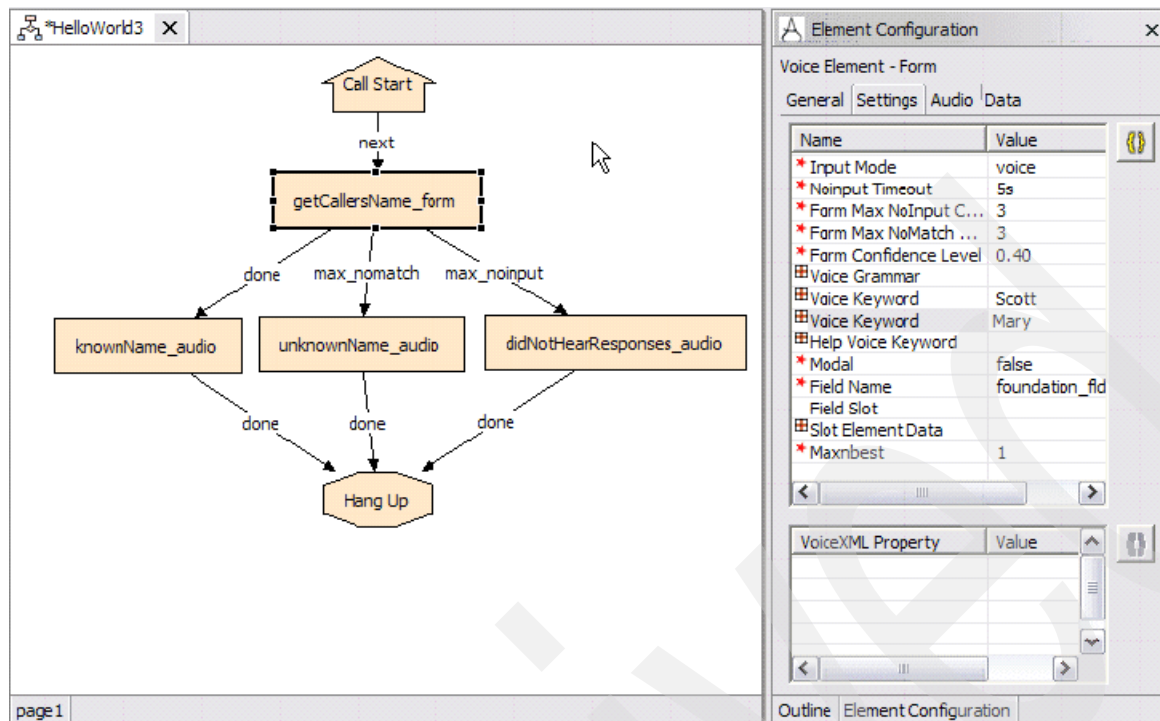


Figure 4-137 HelloWorld3: Setting the second voice keyword to “Mary”

When you are finished adding all your keywords (including the help keyword which will result in the help message being played for the user), the settings should look as shown in Figure 4-138.

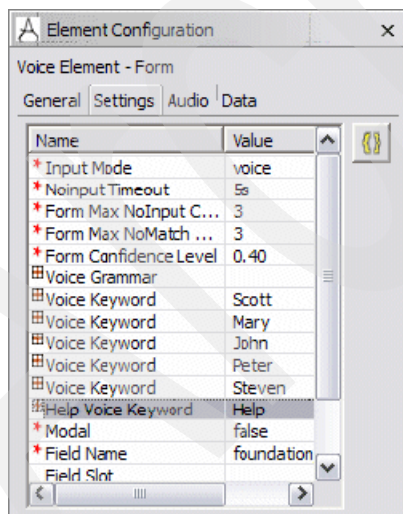


Figure 4-138 Result of adding and setting the Voice Keywords to Scott, Mary, John, Peter, and Steven

- We now have a fully configured form that is ready to accept user input according to well-defined parameters. You can validate, deploy, and test your application. When testing a voice application, employ a rigorous testing methodology. Be sure to simulate conditions that creates real-world use of the application and that ensures that you test all possible execution pathways.



**Troubleshooting:** Here are a couple of things to check if you run into problems during testing.

- ▶ Wondering why you are constantly getting noInput events when you are speaking loudly, clearly, and you are sure your telephone handset or PC microphone is working? Check your project properties and make sure that you have not chosen a DTMF-only VoiceXML gateway in your general settings!
- ▶ While testing, are you noticing that the form seems to recognize names that you did not specify in the grammar? This could be a result of false positives and you might need to boost the recognition confidence threshold for your form. Try increasing it by increments of 0.1 until you stop receiving false positives. Alternately, if your form is not recognizing names that you specified in your grammar, you might want to decrease your confidence threshold slightly. Try decreasing the confidence setting by 0.05-0.1 increments until you get satisfactory results.

### HelloWorld4: add a form with an external grammar

Create a copy of the HelloWorld3 application and call it HelloWorld4. The idea behind HelloWorld4 is to switch from using an inline grammar created in Studio, to an external one. The modifications required to do so are almost trivial. The main point of this exercise is to illustrate the powerful grammar creation toolkit included with IBM Voice Toolkit.

1. Modify our application in studio so that it will use an external grammar instead of the inline one. We added to the getCallersName\_form element. First, remove all the voice keywords we defined for the form element as shown in Figure 4-139.

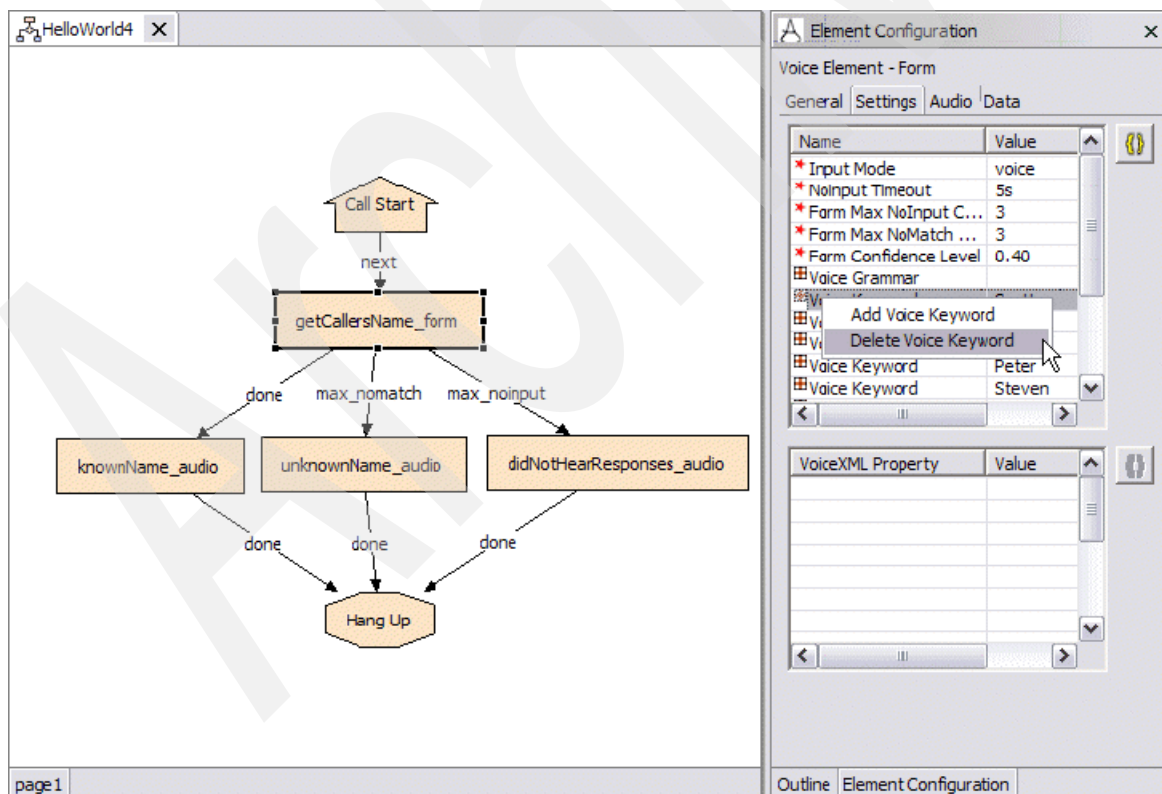


Figure 4-139 HelloWorld4: Deleting a Voice Keyword

When you are finished your form settings should look like Figure 4-140.

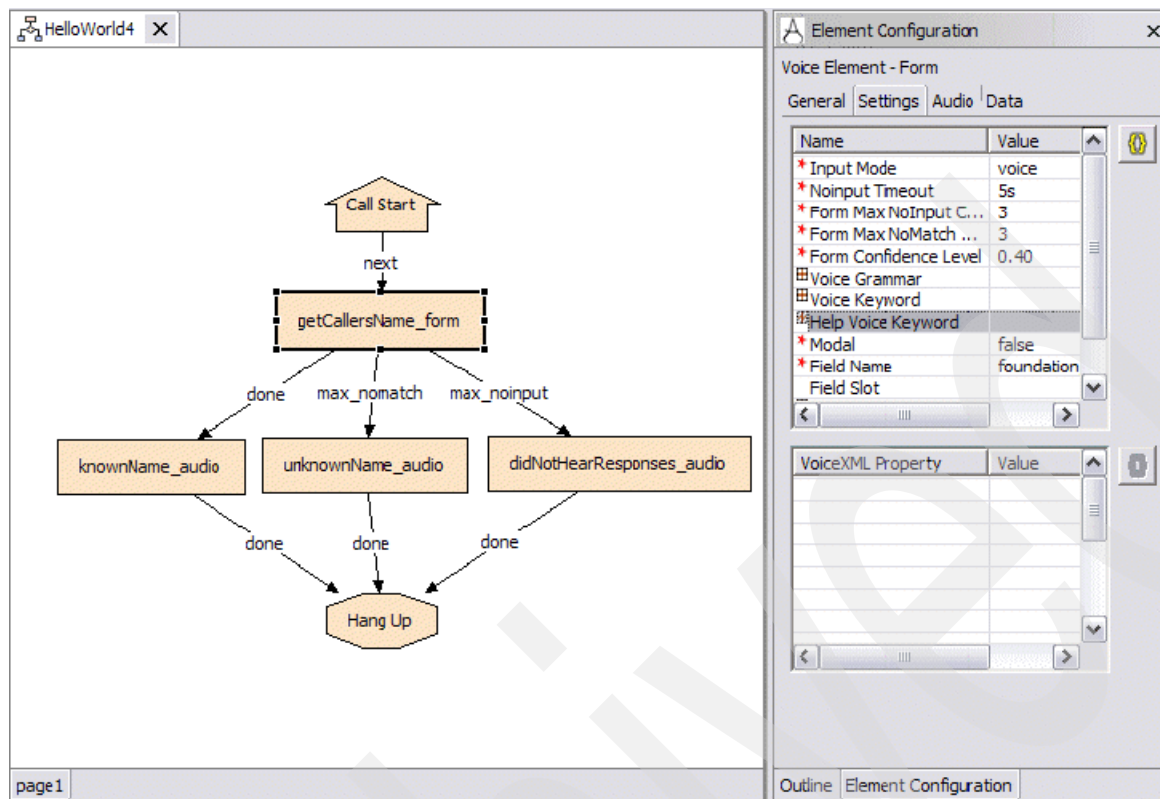


Figure 4-140 HelloWorld4: Result from deleting the voice keywords

2. Define the location for our external grammar that we create shortly. Our grammars and audio resources must be placed in a consistent location, which will be:

<http://9.42.171.24:8080/CVP/grammars/HelloWorld/names.grxml>

Place this URL in the "Voice Grammar" setting for the form as shown in Figure 4-141 on page 216.

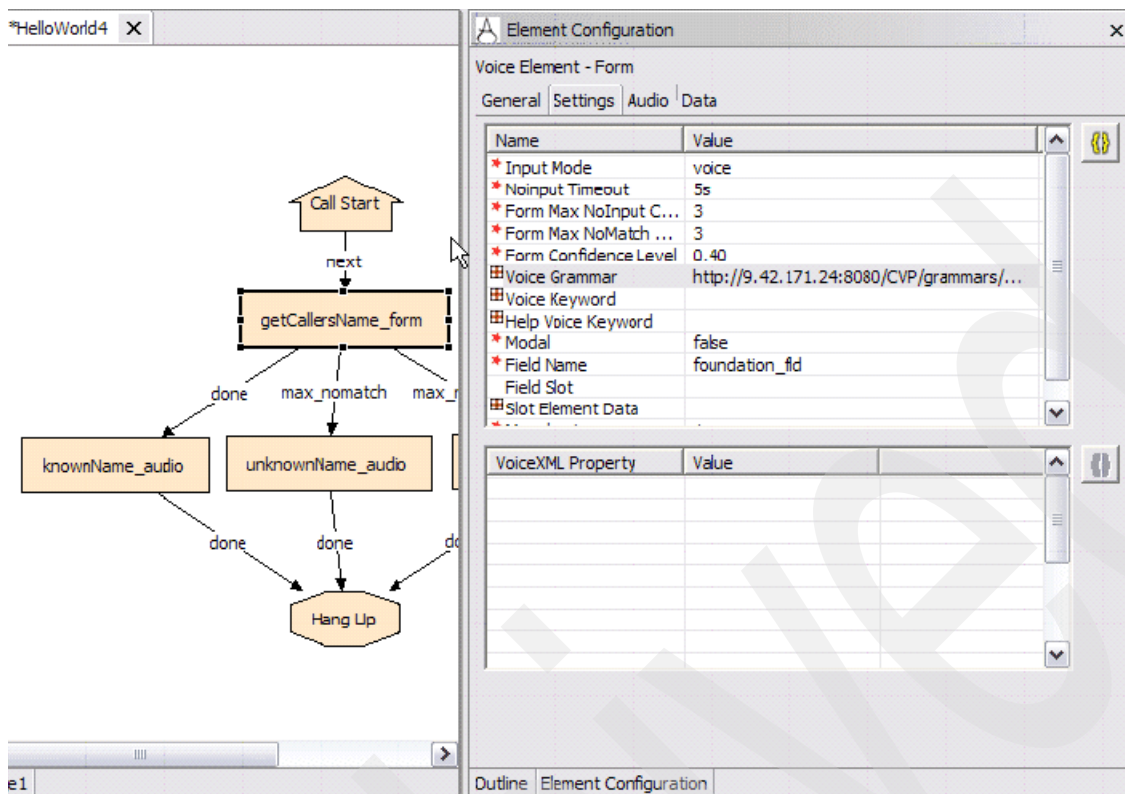


Figure 4-141 HelloWorld4: Setting the Voice Grammar location

This task is complete. Our form now uses an external XML grammar file rather than an inline one.

**Note:** It is possible to use multiple grammars with different weightings and slots and all that other good stuff. For the sake of simplicity, we forego an examination of those concepts, but it is important to know that type of functionality is available should you need it for your applications.

3. Validate, deploy and test.

### Creating an external grammar with IBM Voice Toolkit

This information can also be found in the Voice Toolkit Getting Started Guide. Example 4-5 shows the listing for names.grxml.

Example 4-5 names.grxml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
"http://www.w3.org/TR/speech-grammar/grammar.dtd">
<grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar" tag-format="semantics/1.0"
xml:lang="en-US" mode="voice" root="main_rule">
<rule id="main_rule" scope="public">
  <one-of>
    <item>Scott</item>
    <item>Mary</item>
    <item>John</item>
    <item>Peter</item>
    <item>Steven</item>
  </one-of>
</rule>
</grammar>
```

```

    </one-of>
</rule>
</grammar>

```

---

## HelloWorld5: add a form with confirm

Now, here is where things start to get really interesting. Create a copy of HelloWorld4 and name it HelloWorld5. We now update our application so that it includes a Form\_With\_Confirm element that will confirm what the user has uttered as the name of choice.

1. We delete our getCallersName\_form from the call flow and replace it with a Form\_With\_Confirm element. Rename this element to getCallerName\_formWithConfirm and connect the exit states as shown in Figure 4-142 and Figure 4-143.

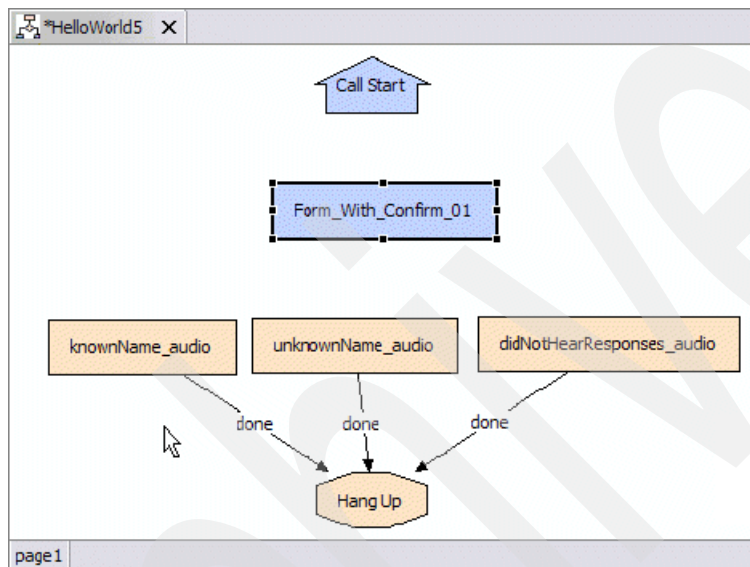


Figure 4-142 HelloWorld5: Adding a form with confirm

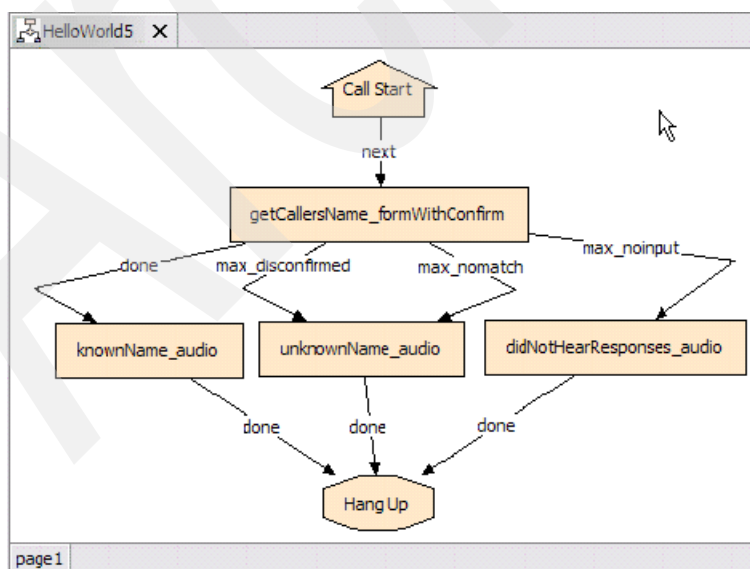


Figure 4-143 HelloWorld5: Connecting the exit states

Notice that we have a new exit state for the Form\_With\_Confirm element: max\_disconfirmed. The max\_disconfirmed exit state is reached simply when the caller is read back his or her utterance and the caller denies the response for the specified maximum number of times. So, for example, if we said Peter and the form said Did you say Peter? and we responded No three times in a row, the max\_disconfirmed exit state would be reached. For the purposes of our application, we assume that a max\_disconfirmed exit state means that we are not correctly understanding the user utterance (a fair assumption, we think) and have connected it to the unknownName\_audio audio element.

2. Configure the settings for our form with confirm (see Table 4-7).

**Note:** There are a couple of settings that are not present in a standard Form element. They are denoted with asterisks.

Table 4-7 Voice Element - Form\_With\_Confirm Settings

Setting Name	Meaning	Value for our application
Input Mode	The type of entry allowed for input. Possible values are: voice   dtmf   both.	voice
Noinput Timeout	The maximum time length allowed (in seconds) for silence or no dtmf entry before a noinput event is thrown.	Leave it at the default value of 5 seconds
Form Max NoInput	0 = infinite noinputs allowed.	We will allow for 3 no input events before we give up
Form Max NoMatch	0 = infinite nomatch allowed.	We will allow for 3 no match events before we give up
Form Confidence Level	The confidence level threshold to use for data capture. This value will be returned by the ASR engine for every match it makes.	Leave this at the default value of 0.40, this is a nice safe value that will recognize most utterances without the danger of returning many false positives. However, you might have to tweak and fine-tune this value to obtain satisfactorily precise results.
Confirm Max NoMatch*	The maximum number of noinput events allowed during form input confirmation. 0 = infinite noinputs allowed.	Leave it at the default value of 3
Confirm Max NoInput*	The maximum number of nomatch events allowed during form input confirmation. 0 = infinite nomatches allowed.	Leave it at the default value of 3
Max Disconfirmed Count*	The maximum number of times a caller is allowed to disconfirm a captured input. 0 = infinite disconfirmations allowed.	Leave it at the default value of 3
Help Voice Keyword	The voice keyword that will trigger the playback of the Help prompt for the form	"Help"

Setting Name	Meaning	Value for our application
Voice Keyword	Inline grammar elements	We will define our inline grammar in the same way we did in HelloWorld3. Add the following voice keywords: "Scott", "Mary", "John" "Peter" or "Steven"

When you have finished your form\_with\_confirm settings, it should look similar to Figure 4-144 on page 219.

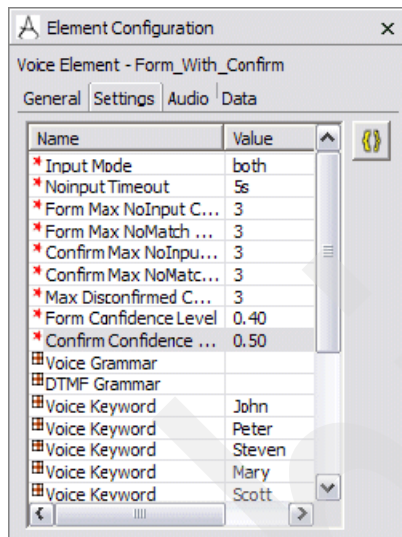


Figure 4-144 Completed Voice Element - Form\_With\_Confirm Settings

3. Configure your Form\_With\_Confirm element with the audio settings show in Table 4-8 (remember to disable barge ins for all audio groups).

**Note:** There are new settings here (denoted with asterisk). The meanings of the new audio groups should be fairly intuitive. They have a similar function as the ones used in data capture, only these are used in the confirmation process instead.

Table 4-8 Form\_With\_Confirm audio settings

Audio Group Name	Found Under	Purpose	Our TTS Text
Form NoMatch	Form Data Capture	This audio group will be played each time the form encounters a noMatch event. We can add several of these so that a different prompt of increasing severity is played for each consecutive event.	noMatch1: "I am sorry I did not understand what you just said. Please try again"  noMatch2: "I am sorry I did not understand what you just said. Please try again, speak clearly please"  noMatch3: "I am sorry that was the third time I did not understand what you said"
Form NoInput	Form Data Capture	This audio group will be played if no input is detected by the form. We can add several of these as well.	noInput1: "I am sorry I did not hear you say anything"  noInput2: "I am sorry I did not hear you say anything for the second time"  noInput1: "I am sorry I did not hear you say anything for the third time."
Form Help	Form Data Capture	This audio group will be played when the user utters the specified help keyword. This prompt should include instructions on how to respond to the question posed by the form.	"Please say the first name of a person. If I recognize this name I will say so."
Add Done	End	This audio group will be played when the data capture process is completed successfully (i.e. before exiting with the "Done" exit state)	"Thank you"
Confirm NoMatch*	Form Data Confirm	Played when no input is received in response to the forms confirmation prompt (Confirm Initial)	"I am sorry I did not understand what you just said."



Audio Group Name	Found Under	Purpose	Our TTS Text
Confirm NoInput*	Form Data Confirm	Played when response to the confirmation prompt (Confirm Initial) does not match an entry in our confirmation grammar	"I am sorry I did not hear what you just said"
Confirm Help*	Form Data Confirm	This audio group will be played when the user utters the specified help keyword while in the confirmation process. This prompt should include instructions on how to confirm input.	"Please confirm whether or not you just said the value that was just read back to you"
Disconfirmed*	Form Data Confirm	This message will be read when the user disconfirms their input.	"Thank you"
Form Initial	Form Data Capture	This is the message that will be read to the caller when the form is first visited	"Hello world. Please say a name"
Confirm Initial	Form Data Confirm	This is where we will read back the captured user utterance. We will need to introduce some new concepts to accomplish this goal.	***To be discussed shortly***

Your audio settings should look similar to Figure 4-145 on page 221.

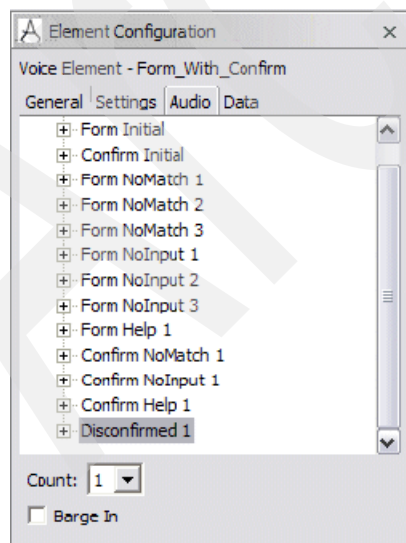


Figure 4-145 Voice Element - Form\_With\_Confirm Settings

## Now we are starting to use element data!

In order for us to enable meaningful confirmation functionality for this form, we have to find a way for the form to read back a TTS string that is not statically defined when we design the application. In other words, we must read back information to the user that is only available at runtime. This is the reason for the existence of variable element data which can be accessed and placed into TTS strings by means of substitution tags. Every element creates a defined group of element data. For example, a Form element defines a piece of element data called Value. The Value variable represents the user utterance that was recognized by the ASR engine and returned to the form. The Value variable is present in all voice elements and allows us to accomplish almost all imaginable coding objectives in a voice application. Now, we have the goal of reading back the user's utterance for the purposes of confirmation in our HelloWorld 5 application. So, what we have to do is to find a way to read this back in the Confirm Initial audio group. This is accomplished in the following way:

1. First we define the static portion of our confirmation message. Suppose we start each confirmation with the phrase "Did you say the name ..."(see Figure 4-146). The idea is that if, for example, the user said the name Mary, the confirmation portion of the form will prompt the user with "Did you say the name Mary?"

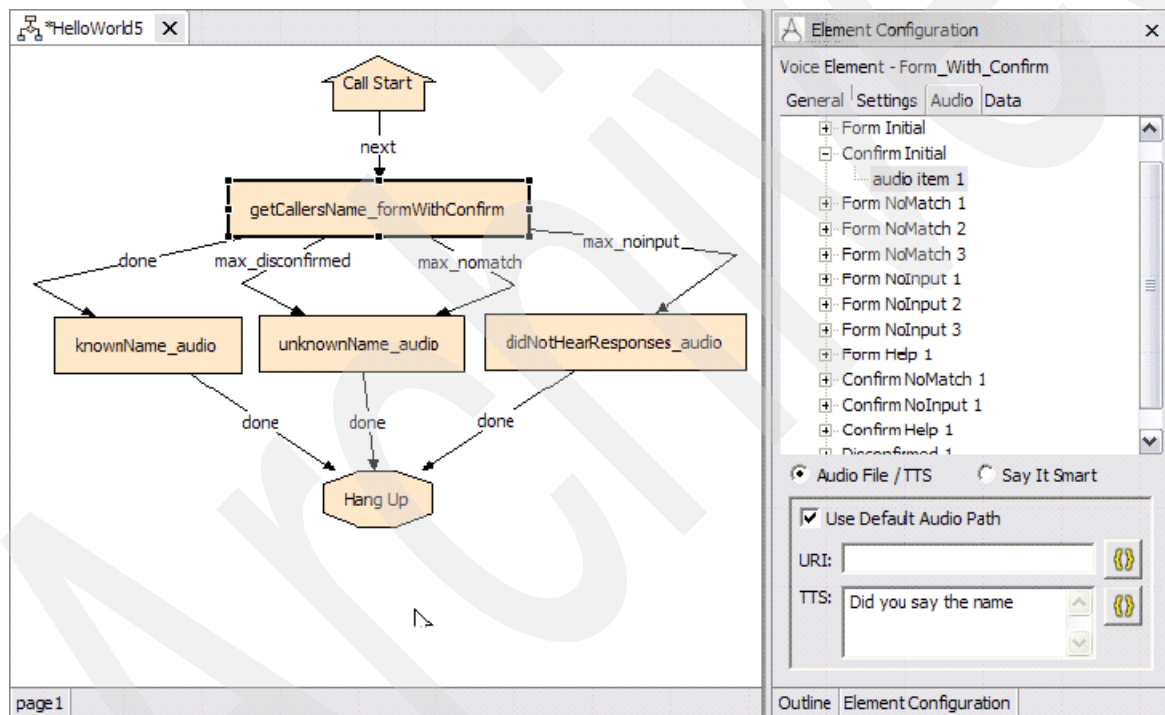


Figure 4-146 HelloWorld5: confirmation message

2. Add another audio item to our Confirm Initial audio group. This audio item will contain the substitution tag that accesses the value variable for our form\_with\_confirm (it will read back the name uttered by the user). First add the audio item to the group as shown in Figure 4-147 and Figure 4-148 on page 223.

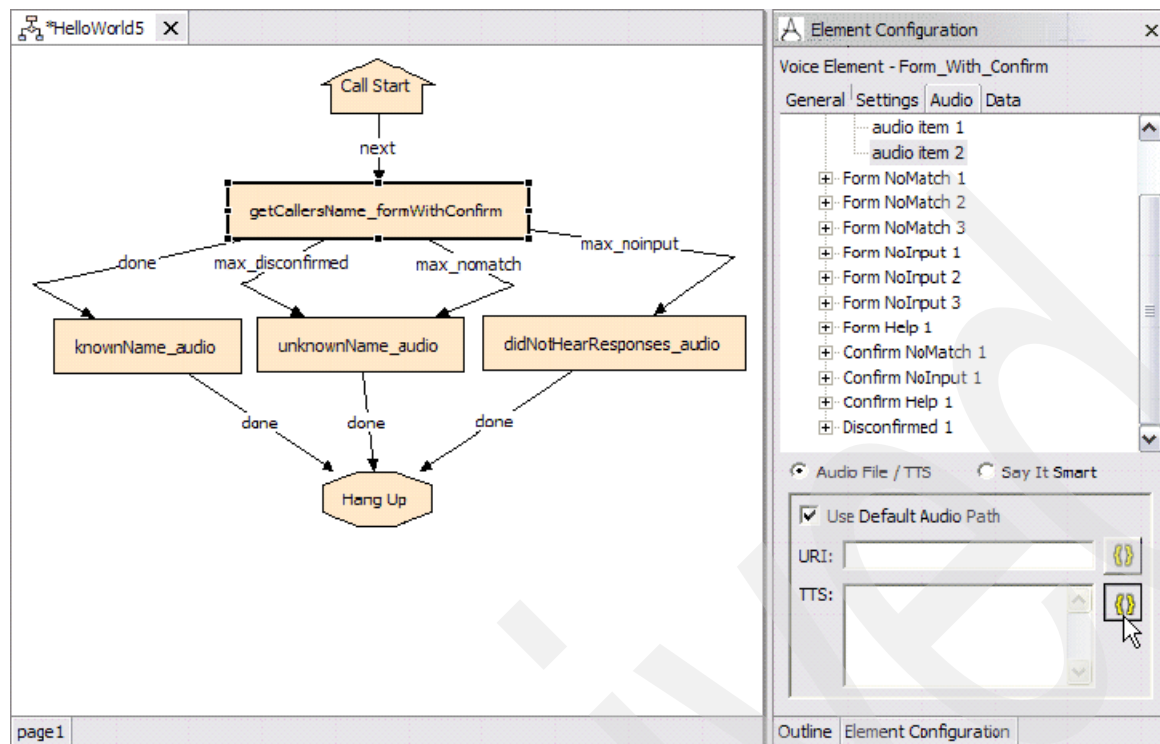


Figure 4-147 HelloWorld5: Adding additional audio item to hold substitution tag

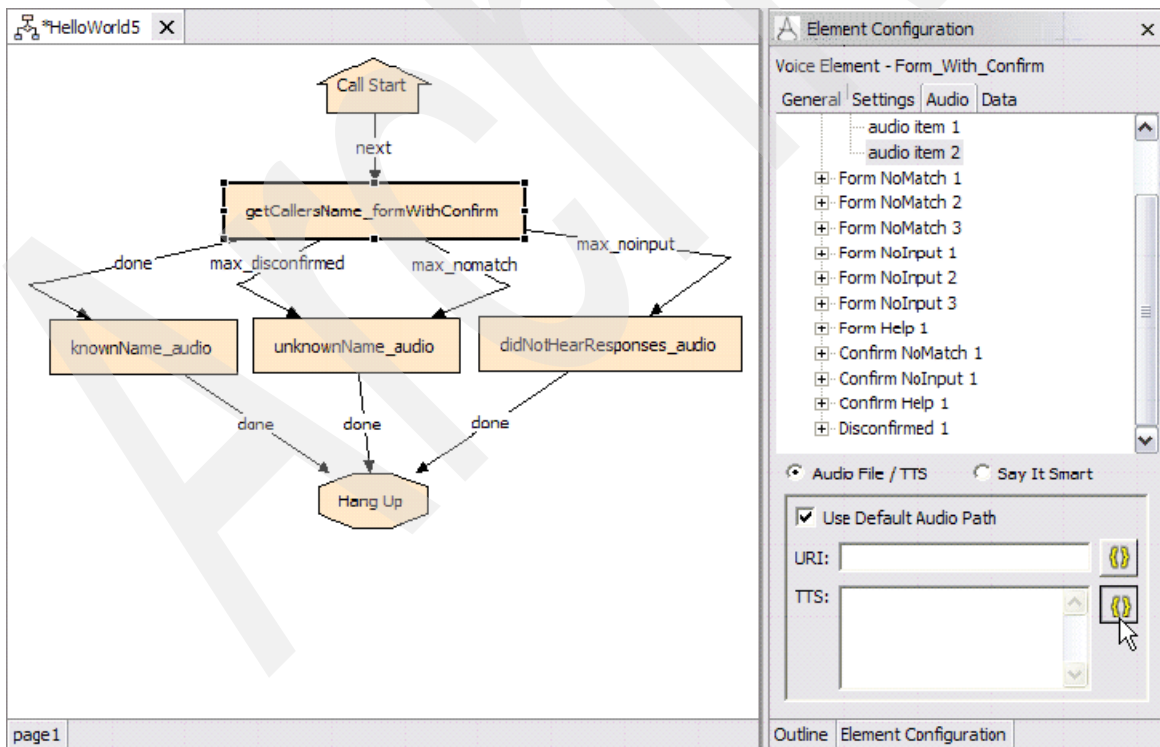



Figure 4-148 HelloWorld5: Opening the Substitution Tag Builder dialog

- Now we define the TTs content for this newly added audio item. To do this, we use the substitution tag builder.

Click the  button to access the substitution tag builder window. You should see the substitution tag builder window as shown in Figure 4-149.

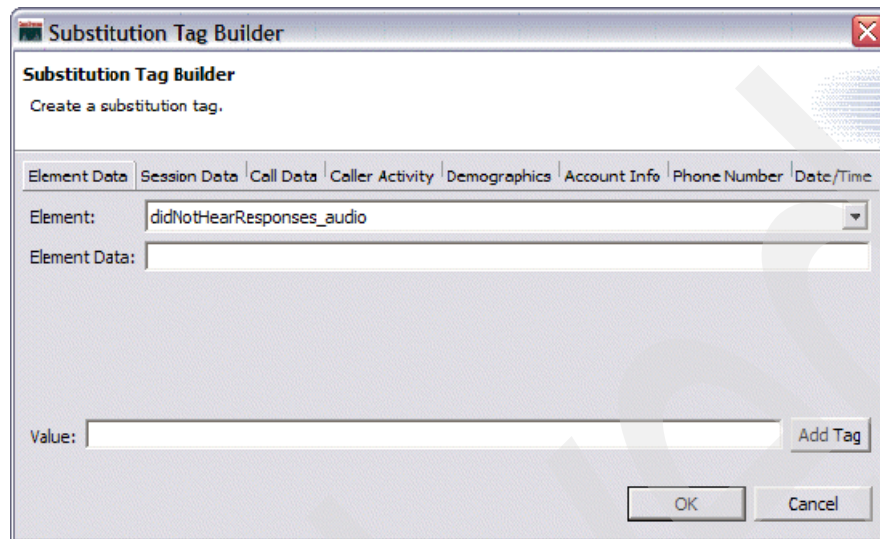


Figure 4-149 Substitution Tag Builder dialog

4. The purpose of this interface is to build a substitution tag that will access the value element variable from the getCallersName\_formWithConfirm element. So the first thing we will need to do is select the appropriate element.

**Note:** The substitution tag builder will allow you to build substitution tags that can access any type of data present in a voice application including: session data, call data (ANI, DNIS, and so on), Demographic information, Account Info, Phone Number and Date/Time information.

5. Select getCallersName\_formWithConfirm from the drop down menu as shown in Figure 4-150.

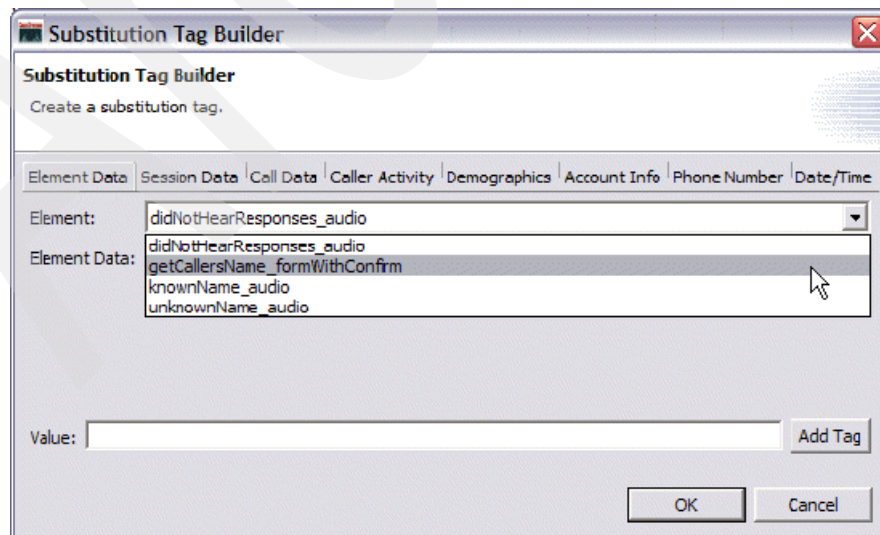


Figure 4-150 Substitution Tag Builder dialog - selecting getCallersName\_formWithConfirm

- Specify which piece of element data we want to access from the `getCallerName_formWithConfirm` element, this is `value`. Type this into the element data field as shown in Figure 4-151 on page 225.

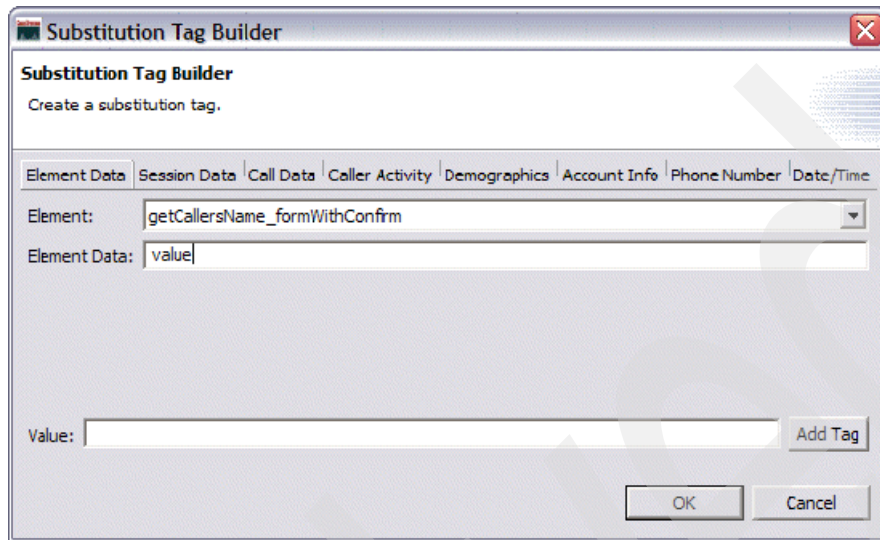


Figure 4-151 Substitution Tag Builder dialog box with “value” in the Element Data field

- Click **Add Tag** to create the substitution tag as shown in Figure 4-152 on page 225. It shows up in the value field.

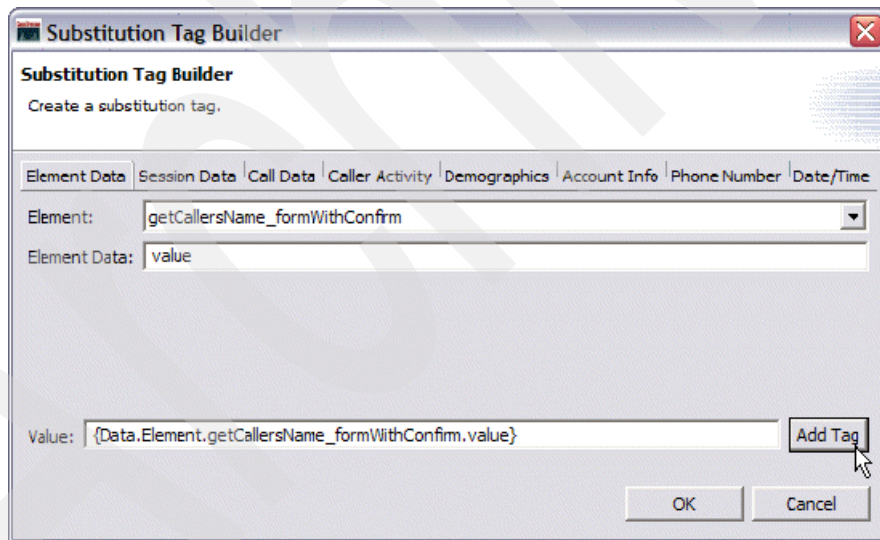


Figure 4-152 Clicking Add Tag to create the substitution tag

- Finally, click **OK**.

You can see that the TTS string for the second audio item of the Confirm Initial audio group is now represented by the substitution tag:

```
{Data.Element.didNotHearResponses_audio.value}
```



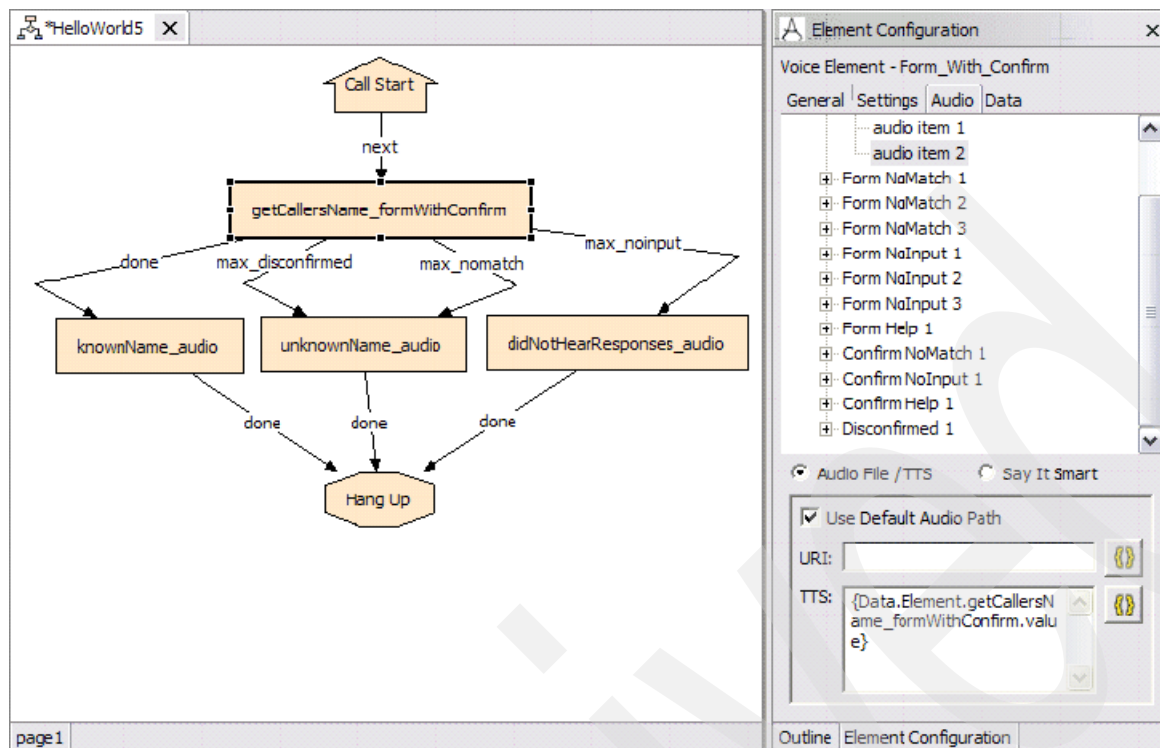


Figure 4-153 HelloWorld5: Result of adding a substitution tag

This simply means that this TTs string will be populated with whatever information is contained in the value element data for the getCallersName\_formWithConfirm element at runtime.

9. We can now validate and deploy our application. Go ahead and test it. It is clear from this example just what a powerful concept substitution tags are.

Table 4-9 shows a fairly comprehensive listing of substitution tags available to the developer, which is taken directly from CVP VoiceXML Users Guide.

Table 4-9 Substitution tags and their descriptions

Tag Content	Description
CallData.ANI	The ANI of the current call or "NA" if not sent.
CallData.DNIS	The DNIS of the current call or "NA" if not sent.
CallData.UUI	The UUI of the current call or "NA" if not sent.
CallData.IIDIGITS	The IIDIGITS of the current call or "NA" if not sent.
CallData.SOURCE	The name of the application that transferred to this one.
CallData.APP_NAME	The name of the current application.
CallData.DURATION	The duration, in seconds, of the call up to this point.

Tag Content	Description
Data.Session. <u>VAR</u>	The value of Session Data where VAR represents the name of the session variable. The object stored there will be represented as a string.
Data.Element. <u>ELEMENT.VAR</u>	The value of Element Data where ELEMENT represents the name of the element and VAR represents the name of the element variable.
CallerActivity.NthElement. <u>N</u>	The name of a certain element visited in the call where N represents the number for the nth element.
CallerActivity.NthExitState. <u>N</u>	The name of a certain element's exit state visited in the call where N represents the number for the nth element.
CallerActivity.TimesElementVisited. ELEMENT	The number of times an element was visited in the call where ELEMENT represents the name of the element.
CallerActivity.TimesElementVisitedExitState. <u>ELEMENT.EXIT_STATE</u>	The number of times an element was visited in the call with a particular exit state where ELEMENT is the name of the element and EXIT_STATE is the exit state.
GeneralDateTime.HourOfDay.CURRENT	The current hour.
GeneralDateTime.HourOfDay.CALL_START	The hour the call started.
GeneralDateTime.Minute.CURRENT	The current minute.
GeneralDateTime.Minute.CALL_START	The minute the call started.
GeneralDateTime.DayOfMonth.CURRENT	The current day of the month.
GeneralDateTime.DayOfMonth.CALL_START	The day of the month the call started.
GeneralDateTime.Month.CURRENT	The current month.
GeneralDateTime.Month.CALL_START	The month the call started.
GeneralDateTime.DayOfWeek.CURRENT	The current day of the week.
GeneralDateTime.DayOfWeek.CALL_START	The day of the week the call started.
GeneralDateTime.Year.CURRENT	The current year.
GeneralDateTime.Year.CALL_START	The year the call started.



**Note:** Keep these concepts in mind when working with substitution tags.

- ▶ Each Date / Time tag evaluates to 0-23 when referring to the hour, 0-59 when referring to the minute, 1-12 when referring to the month, 1-31 when referring to the day of the month, 1-7 when referring to the day of the week (where 1 is Sunday), and the year is represented as a four-digit number.
- ▶ If any data represented by the tag ends up as null, substitution will render it as an empty string. For example, if a setting contained “source{CallData.SOURCE}” and there was no application that transferred to the current application, the setting would be evaluated as “source”. In this case, a warning appears in the Error Log for the application noting that a substitution value was null and was replaced with an empty string.

### **HelloWorld6: store the confirmed result in session data**

In the previous example, we saw the power of element data and how to access element data using substitution tags. Create a copy of HelloWorld5 and call it HelloWorld6. We take this example one step further and store the confirmed value from our `getCallerName_formWithConfirm` in session data and then read it back to the user from the `knownName_audio` audio item found after the `form_with_confirm` in the call flow.

Elements have the ability to create session variables. This is a handy way to stow information away that can be altered by any element in the call flow (remember that element data has the limitation of only being able to be altered by the element that created it in the first place, this makes session variables a somewhat more flexible repository for various pieces of information that may have to be manipulated at various points in the call flow) This can be accomplished in the element configuration in the following way:

1. Go to the **Data** tab for the `getCallersName_formWithConfirm` configuration and specify a name for the session variable that you would like to store the users name in. Call it `nameSessionVariable` (see Figure 4-154 on page 229).

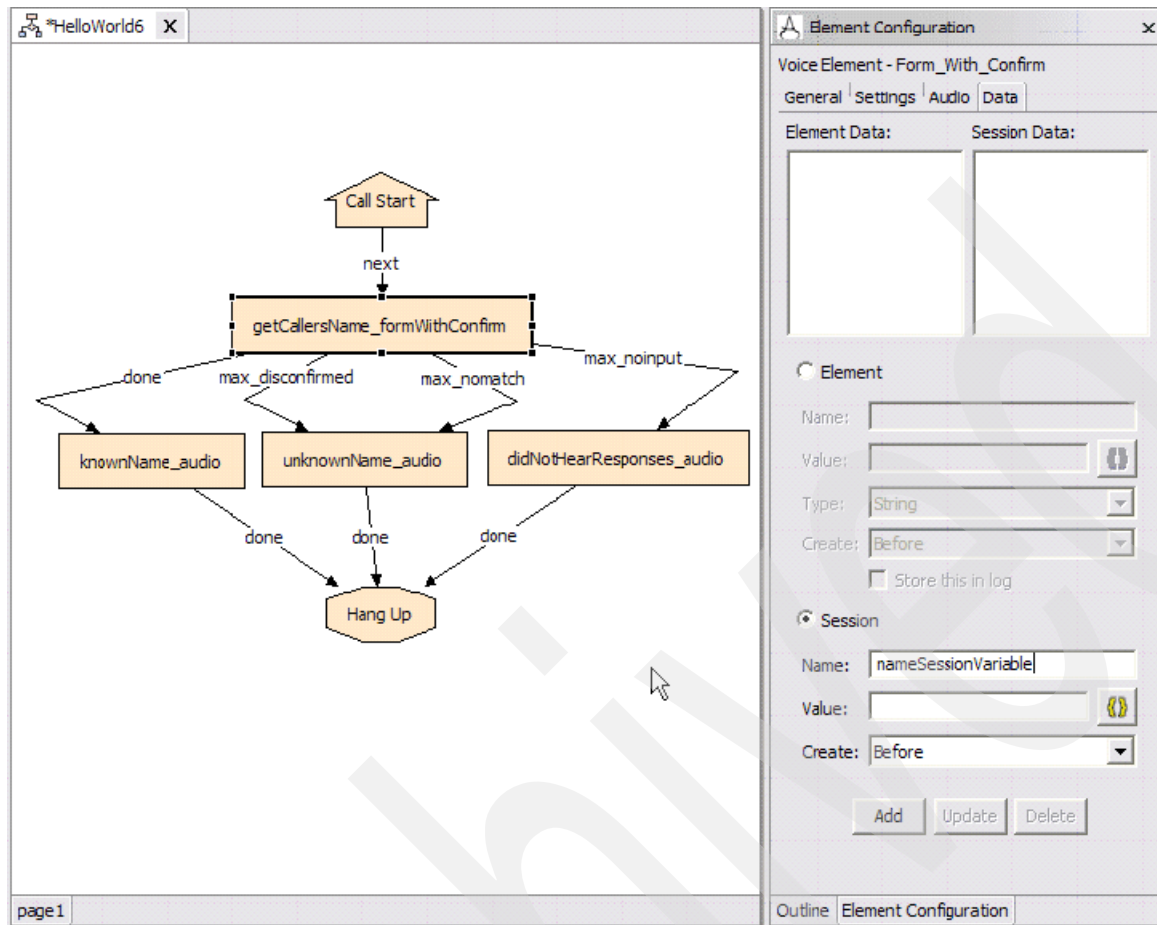


Figure 4-154 HelloWorld6: session data

2. We assign this variable the value from the getCallersName\_formWithConfirm value element data. We do this by creating a substitution tag (“{Data.Element.getCallersName\_formWithConfirm.value}”) just as we did in HelloWorld5. The session variable will now be assigned whatever value is contained in the element data at runtime (see Figure 4-155 on page 230).

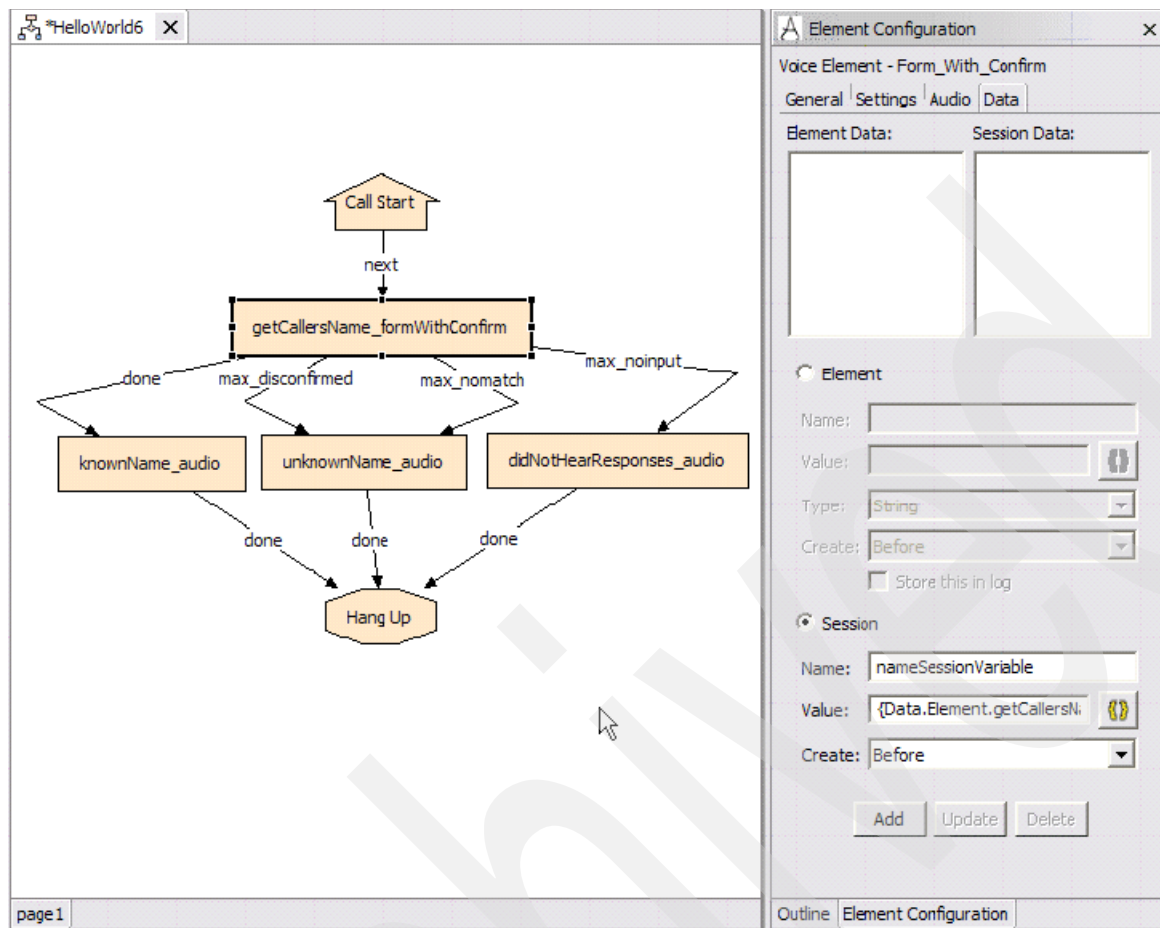


Figure 4-155 HelloWorld6: associating a substitution tag with a session data variable

3. Finally, we specify that this session variable be created immediately After this element is visited (see Figure 4-156 on page 231). Why? Well, simply because the element data for the form will be empty until the user has specified a value and confirmed it. If we create this variable before the element is visited, it will simply hold a null value, which is not terribly useful.

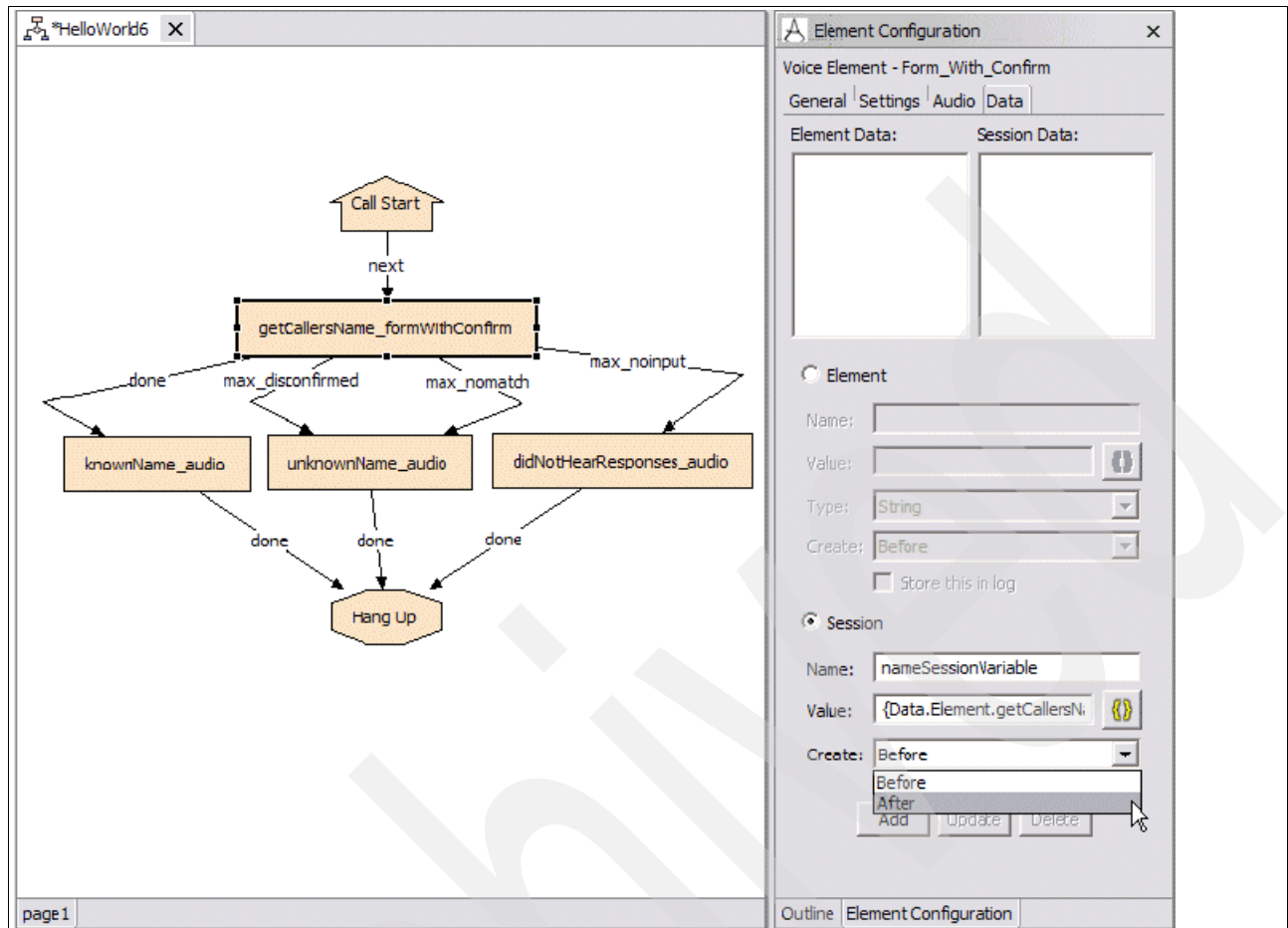


Figure 4-156 HelloWorld6: create the session variable after the element is visited

4. Click **Add** to add this variable to the list of variables that this element will create, as shown in Figure 4-157 on page 232.

**Note:** You can create as many session variables as you like. You can also create as many novel pieces of element data as you wish. Just remember that only the `getCallersName_formWithConfirm` will be able to alter its own element data. Therefore, you should only bother to create new element data for elements that will be visited more than once in the call flow and require some sort of temporary storage mechanism. Otherwise, if you have information that must be accessed and manipulated by multiple elements in the call flow, use session data to store this information.

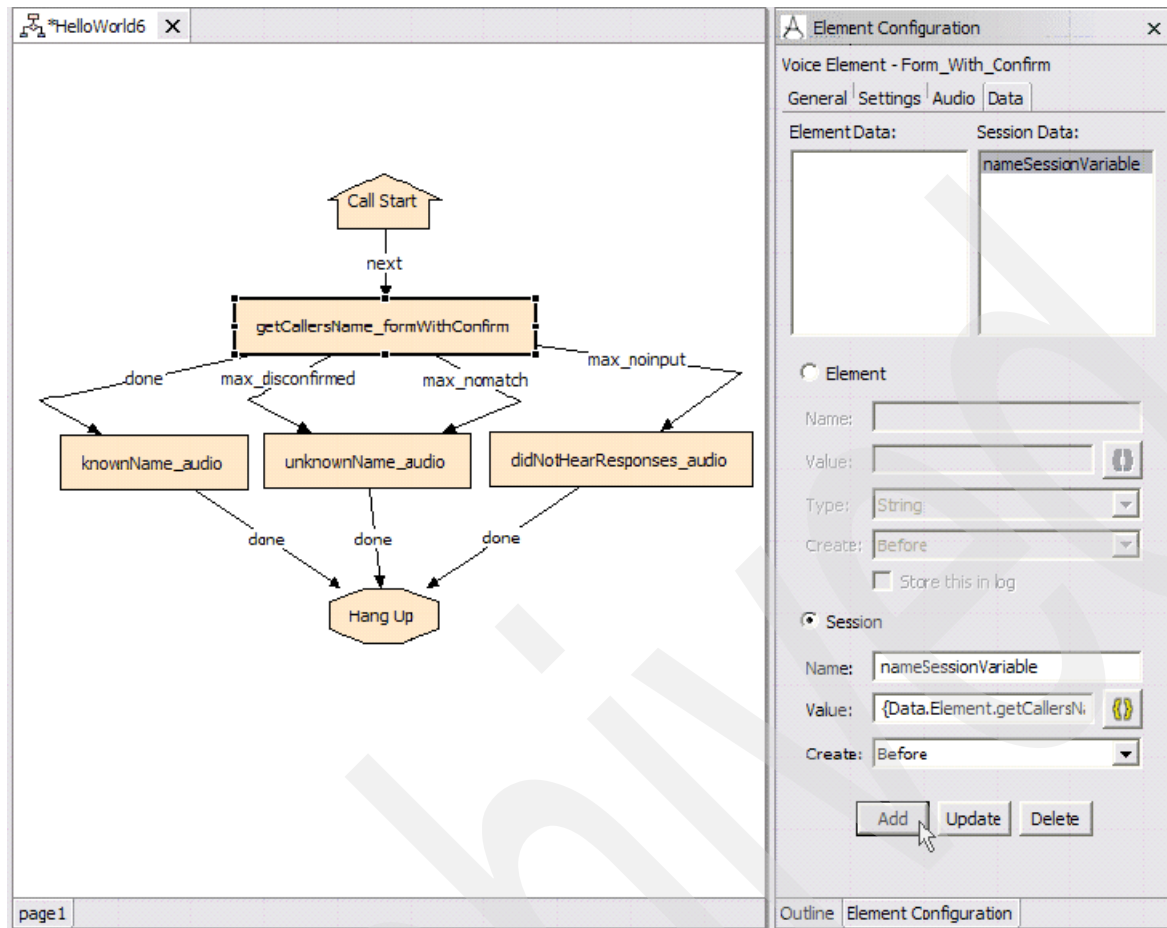


Figure 4-157 HelloWorld6: a new session data variable has been added

We now have a form with confirm element that gets a name from the user, confirms it and then stores this value in the “nameSessionVariable” session variable. Now, we will tweak the configuration of the knownName\_audio element so that it accesses the value in the session variable and reads it back to the user:

1. Add a new audio item to the Initial audio group of the knownName\_audio element. Set the TTS content for this new audio item to {Data.Session.nameSessionVariable} using the substitution tag builder as shown in Figure 4-158 on page 233, Figure 4-159 on page 233, and Figure 4-160 on page 234.

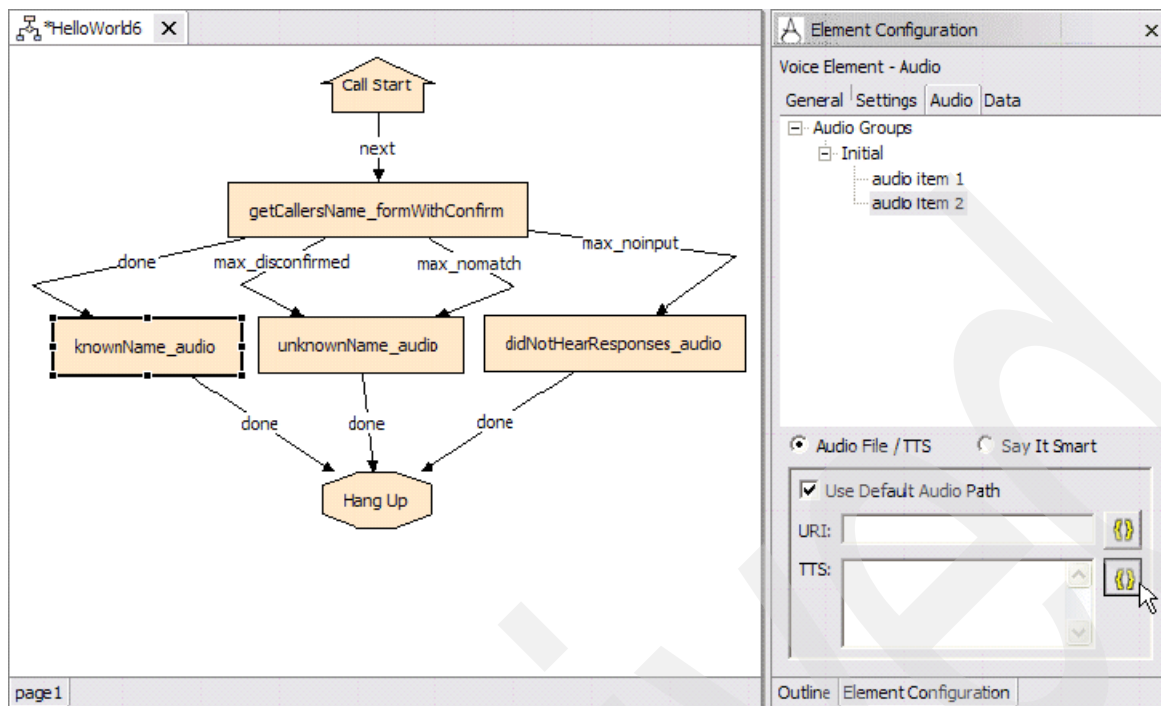


Figure 4-158 HelloWorld6: Using the substitution tag builder to set the TTS content

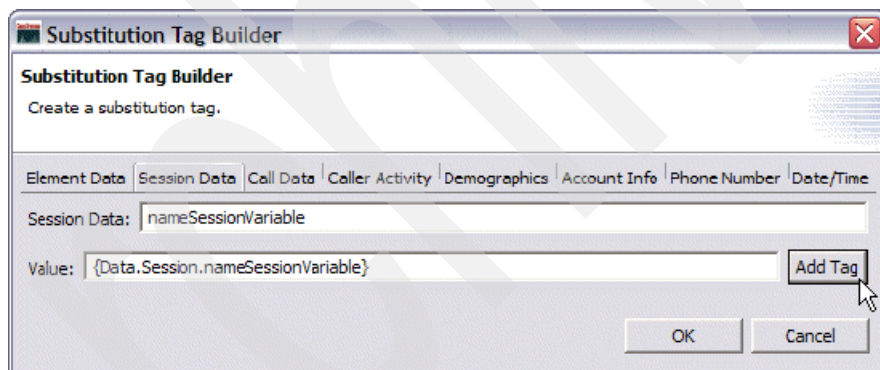


Figure 4-159 Substitution Tag Builder dialog - Entering the Session Data and Value fields

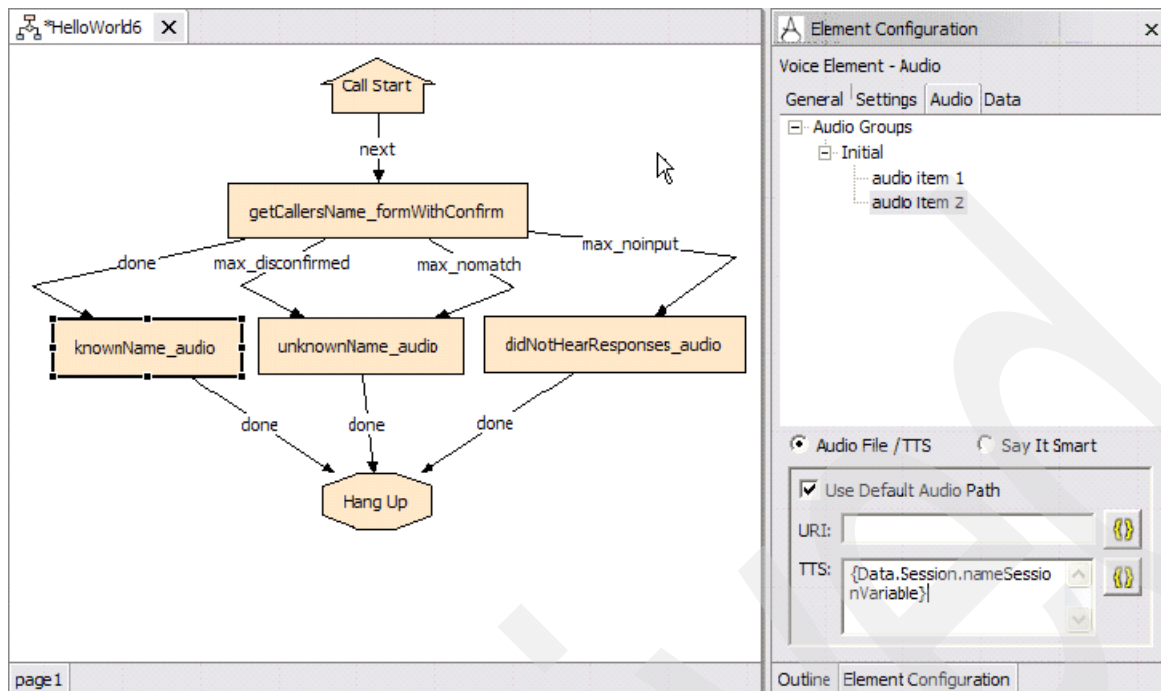


Figure 4-160 HelloWorld6: Result of using the substitution tag builder to set the TTS content

2. Now, go ahead: validate, deploy and test. We have read a user utterance from element data, and also from session data too. Experiment a bit on your own. See what happens when you create the session variable before the element is visited. Play around and explore. This is the best way to get a good feel for how things work in studio.

### HelloWorld7: add a hotlink plus DTMF input

Now we are going to change gears a little bit. In this section, we create an application that illustrates the use of Hotlinks. A *hotlink* was previously defined as: “A globally accessible utterance and / or key press that immediately brings the call to a specific part of the call flow or throws an event.”

This element can be extremely handy to provide zero-out type functionality or global help functionality in larger applications. The way we illustrate the concept is by creating a series of five forms. These forms are just dummy forms that do nothing but say You are in form 1, You are in form 2, and so forth. Each form will have its own individual help in the form of This is help for form 1. There will be a global help that is accessed at any point in the call flow by the user uttering Global. This exercise illustrates the difference between contextual help and global help. This application also uses DTMF input to activate a global help menu. Create a new project call HelloWorld7:

1. Drag a form into the call flow. Call it 1\_form. Set the Form Initial TTS string to You are in form 1. Add a help audio group to the form. Set the help text to This is the help for form 1. Set the Help Keyword for this form to **Contextual**. Add one voice keyword as well: Nothing. Set max noinput and max nomatch counts to 1 each. Please refer to Figure 4-161 on page 235.



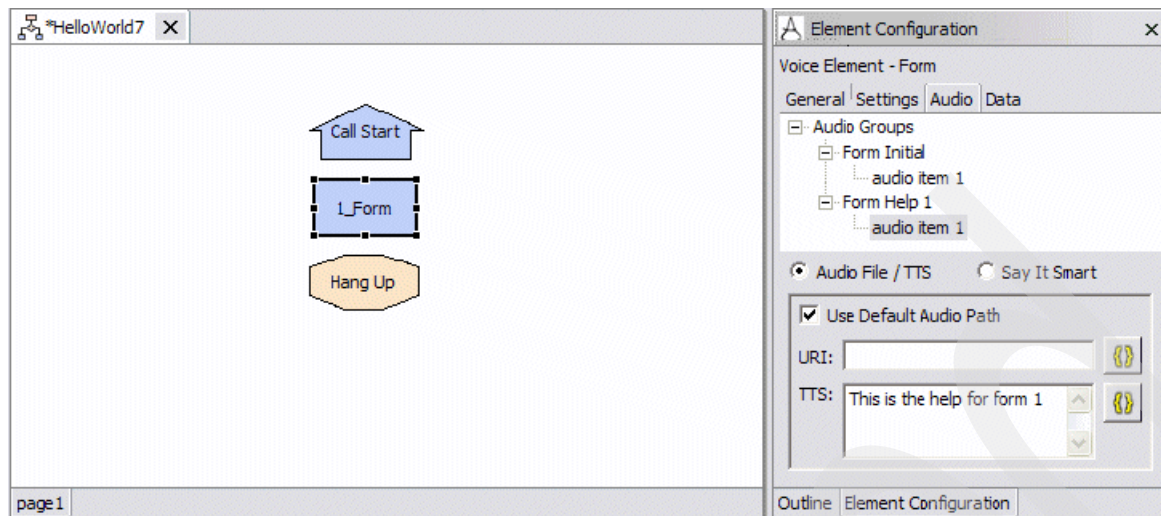


Figure 4-161 HelloWorld7: Creating the initial call flow with 1\_Form

2. Create five copies of this form and link them up as shown in the call flow (see Figure 4-162). Also modify the initial and help audio for each of the copies so that they correspond to the form name, for example, for 3\_form change initial to: You are in form 3 and help to This is the help for form 3.

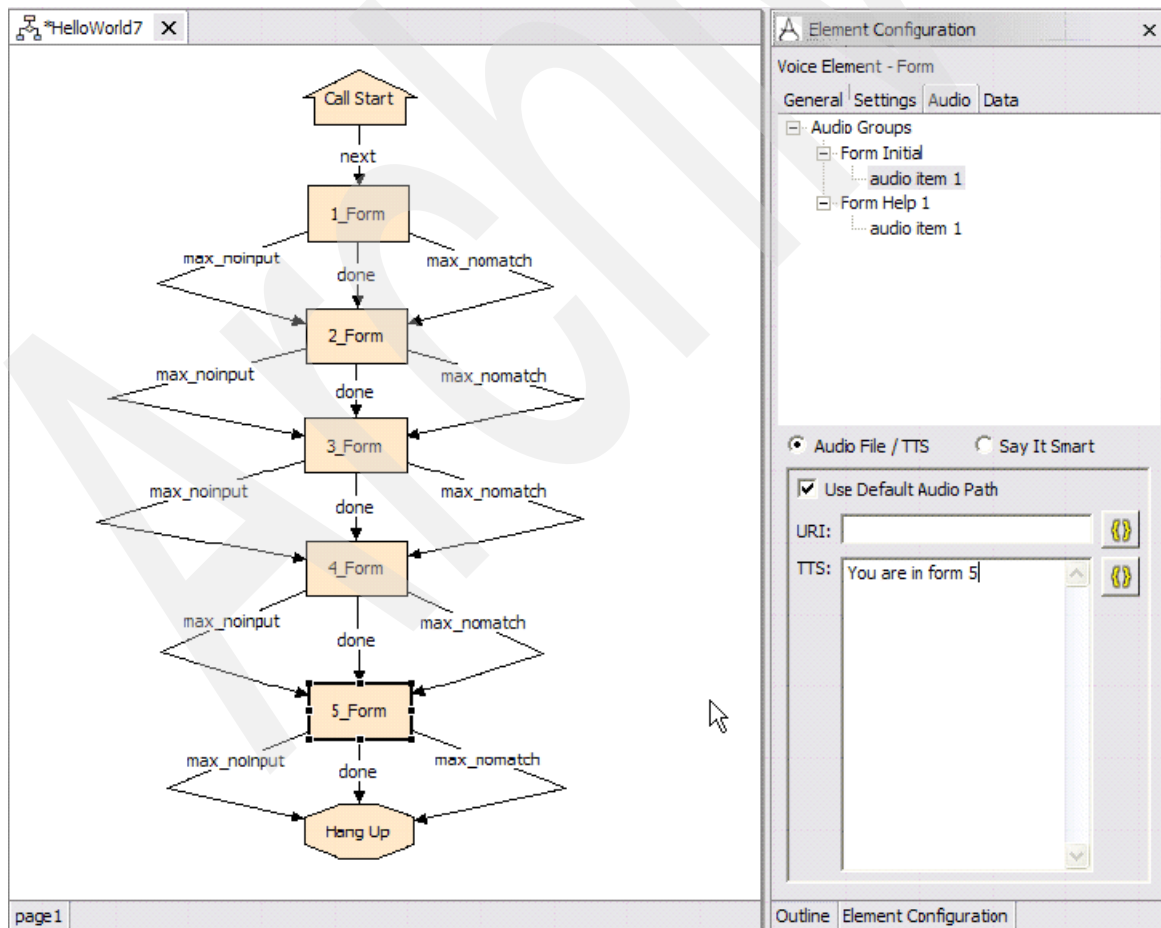


Figure 4-162 HelloWorld7: Duplicating 1\_Form and linking them into the call flow

3. Add a Hotlink element to the call flow as shown in Figure 4-163 on page 236.

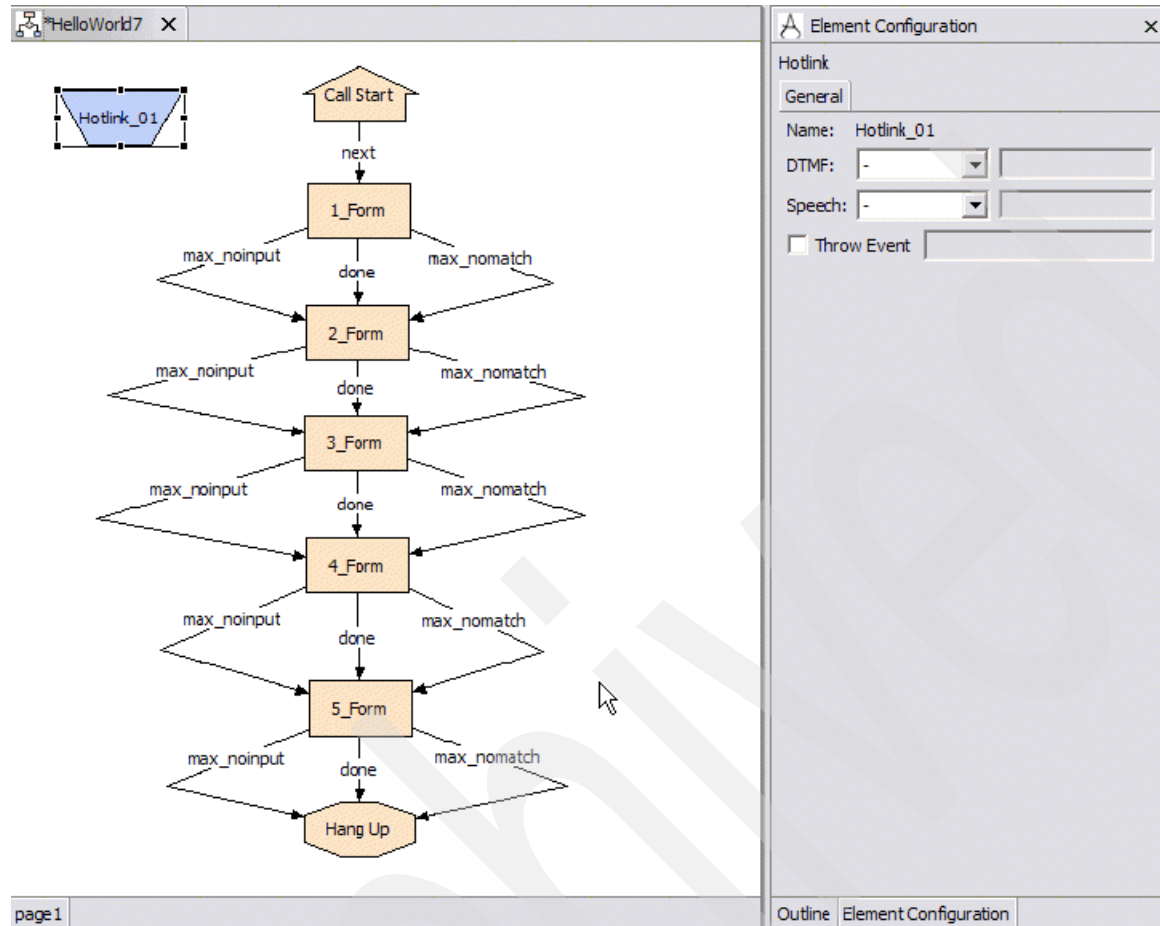


Figure 4-163 HelloWorld7: Adding a Hotlink element to the call flow

4. Configure our hotlink so that it recognizes the keyword Global as shown in Figure 4-164 on page 237.

**Help Tips:** Here are a couple of alternative ways of defining keywords:

- For a more sophisticated set of keywords, use an external grammar file.
- You can also define a global DTMF keypress that will cause this hotlink to be activated. Let us make ours the star ("\*") key.

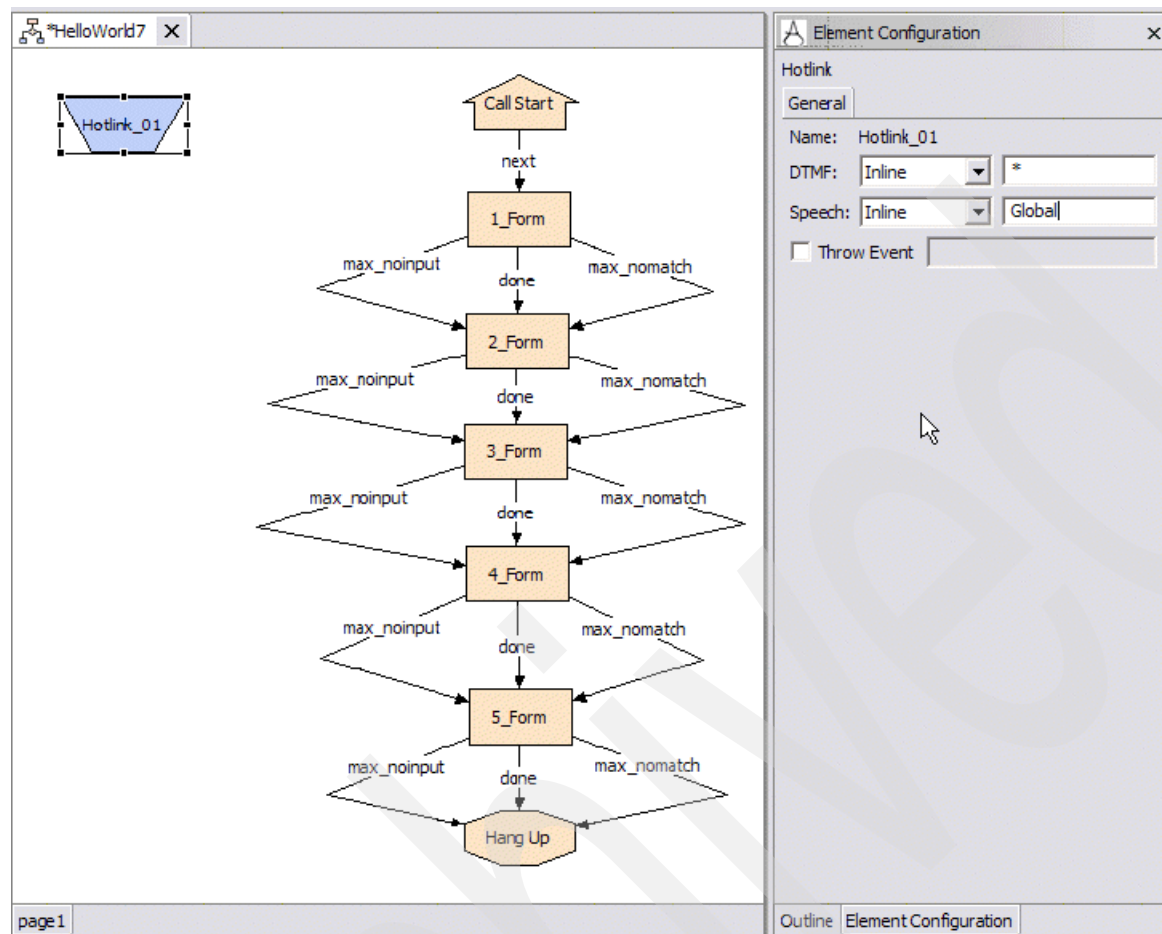


Figure 4-164 HelloWorld7: Configuring the Hotlink as Global and defining a DTMF keypress

5. We must have our hotlink lead to somewhere. Arrange it so it leads to an audio group that reads the TTS string This is the global help. You will now go back to the beginning of the call flow. At that point the TTS string jumps back into the call flow at the beginning. Please refer to Figure 4-165 on page 238.

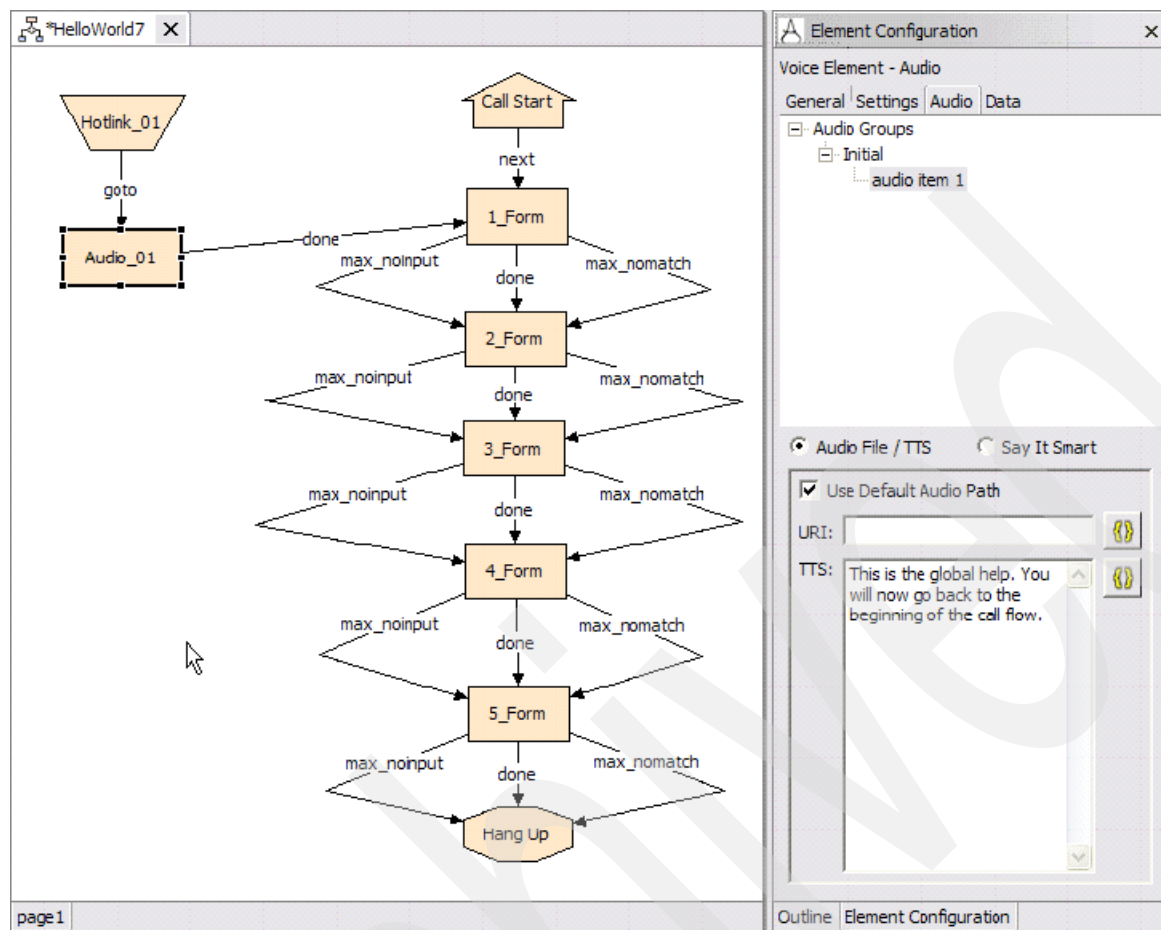


Figure 4-165 HelloWorld7: Connecting the Hotlink to an Audio Group and having it jump back to call flow beginning

#### 6. Validate, deploy and test.

What will happen when you run this application? Well if you just sit there quietly, the application will move through all five forms. If you say the word “Contextual” or press the star key while you are in a form, you will get the help message for the form you are currently in (contextual help). If you utter the word “Global,” you will get the global help message regardless of where you are in the call flow. Then execution will revert back to the beginning of the call flow. Needless to say, you do not want your call to restart immediately after the user requests help. However, we must build a slightly more sophisticated application to reinsert the user into their prior position in the call flow when a hotlink is activated.

### HelloWorld8: add a decision element to implement application flow control

In HelloWorld7, we added a globally accessible hotlink that would read a global help menu and then throw the user back to the beginning of the call flow. That is clearly a very developer-friendly approach to creating global help prompts, but real-world users might find this approach somewhat troublesome. What we want to do now is implement some sort of system for tracking where exactly the user is in the call flow so that if the user triggers the global help menu, they can be placed back into the call flow at their previous location after the global help has been played. The obvious thing to do to implement this functionality would be to use a flag element, However flag elements have one serious limitation. Once they are

tripped, they cannot be untripped. This would not really be a problem in our application because the flow should be completely unidirectional. However, the most robust approach (one that will work in applications where portions of a call flow might be visited more than once) is to use a session variable that tracks exactly where the user is in the call flow and a decision element that branches back to the appropriate location upon termination of the global help menu. Be very careful; we are getting into far more advanced application development from this point forward.

Create a copy of HelloWorld7 and call it HelloWorld8.

1. Modify 1\_Form so that it creates a session variable called `currentPosition`, which has a value of 1 before the element is visited (see Figure 4-166). Now do the same for all other forms in the application. Set the value of the variable to the number of the form. So for 2\_Form, `currentPosition = 2`, for 3\_Form `currentPosition = 3`, and so forth.

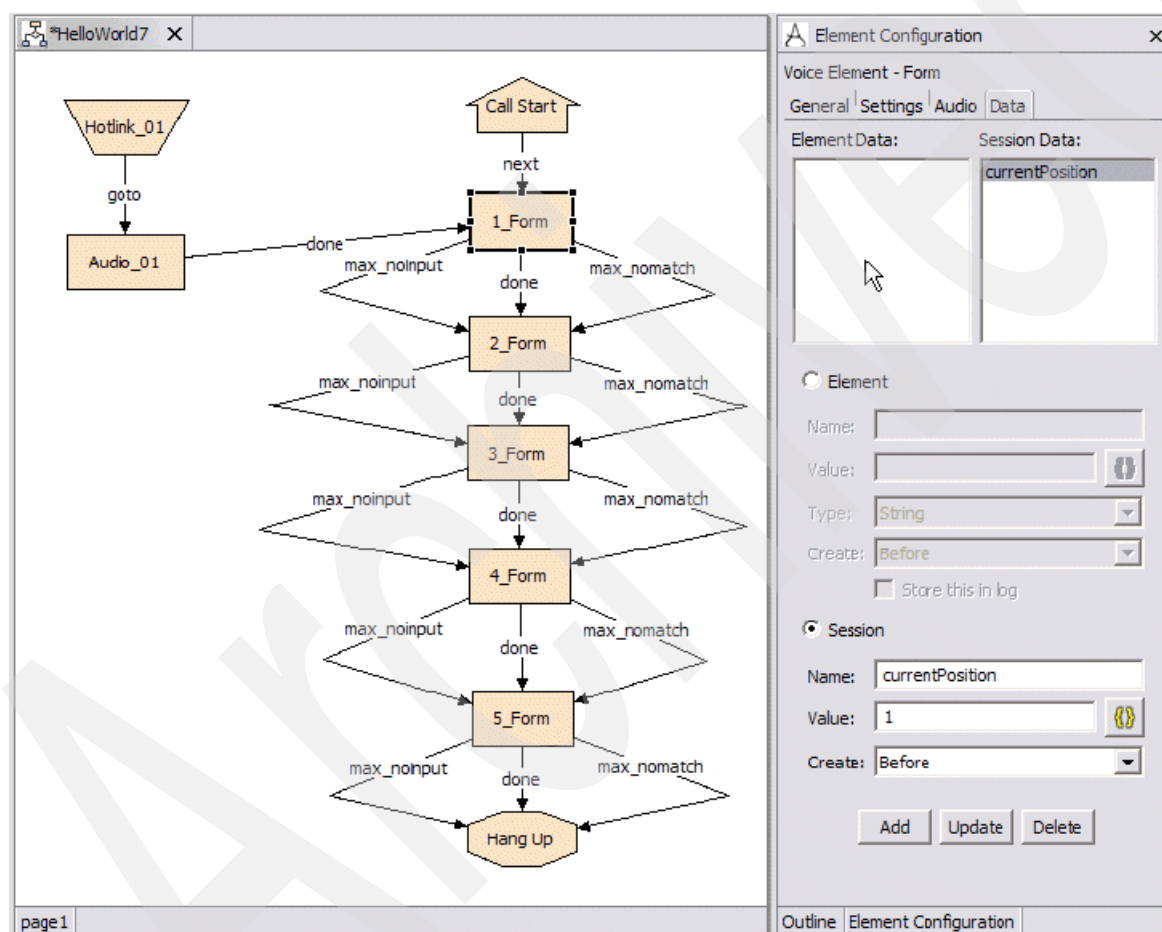


Figure 4-166 HelloWorld7: Create a session variable called `currentPosition`

2. Our application is now configured in a way such that when the application starts, the value in `currentPosition` will be 1. Every time a form is visited, then the `currentPosition` value will be updated to the position of the form in the call flow. This is how we will track where the user is in the application. Disconnect the exit state for `audio_01` and rename it too as shown in Figure 4-167 on page 240.



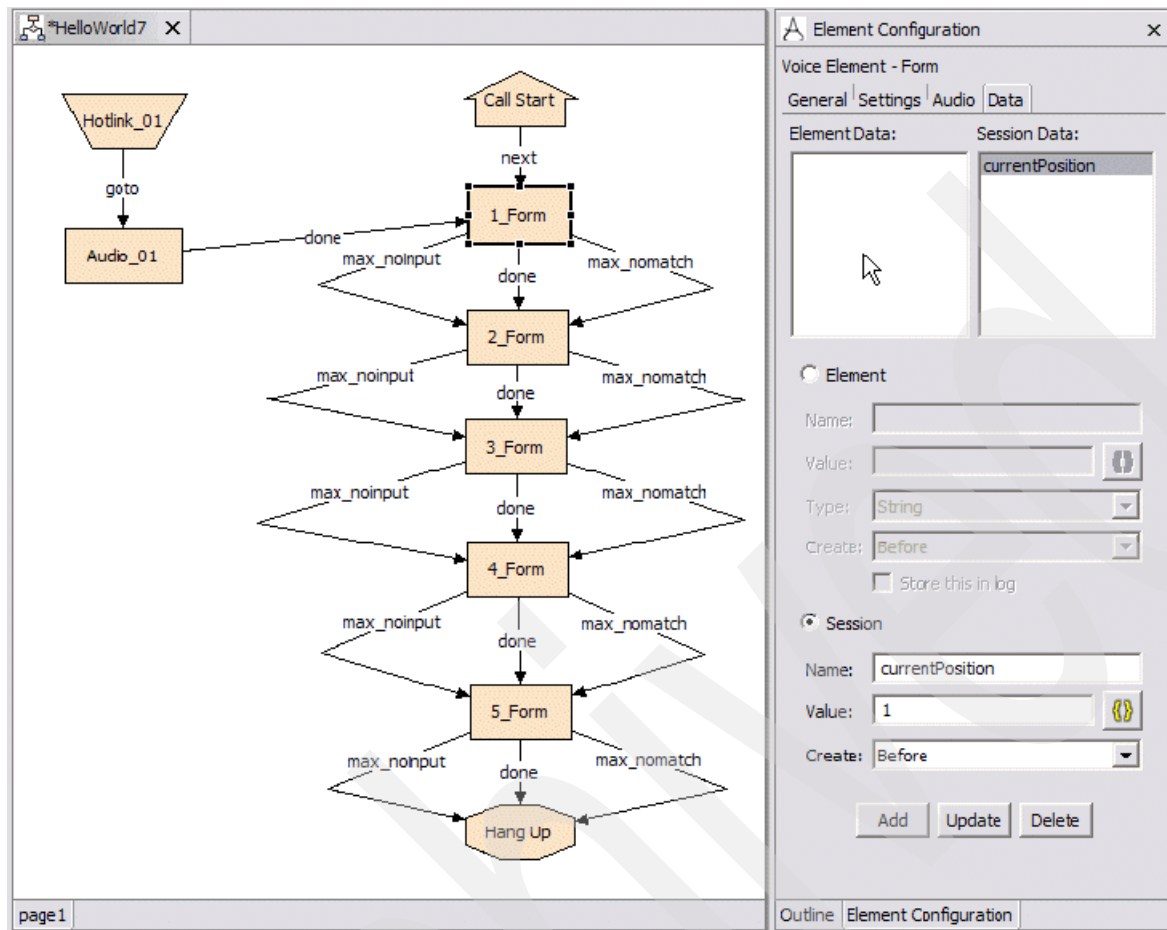


Figure 4-167 HelloWorld7: Renaming Audio\_01 to globalHelp\_audio

3. Add a decision element to the call flow. Call it returnToPreviousPosition\_decision. Connect the audio element to the decision element. Now define five exits states for the decision element: return\_to\_1, return\_to\_2, return\_to\_3, return\_to\_4 and return\_to\_5 as shown in Figure 4-168 on page 241.

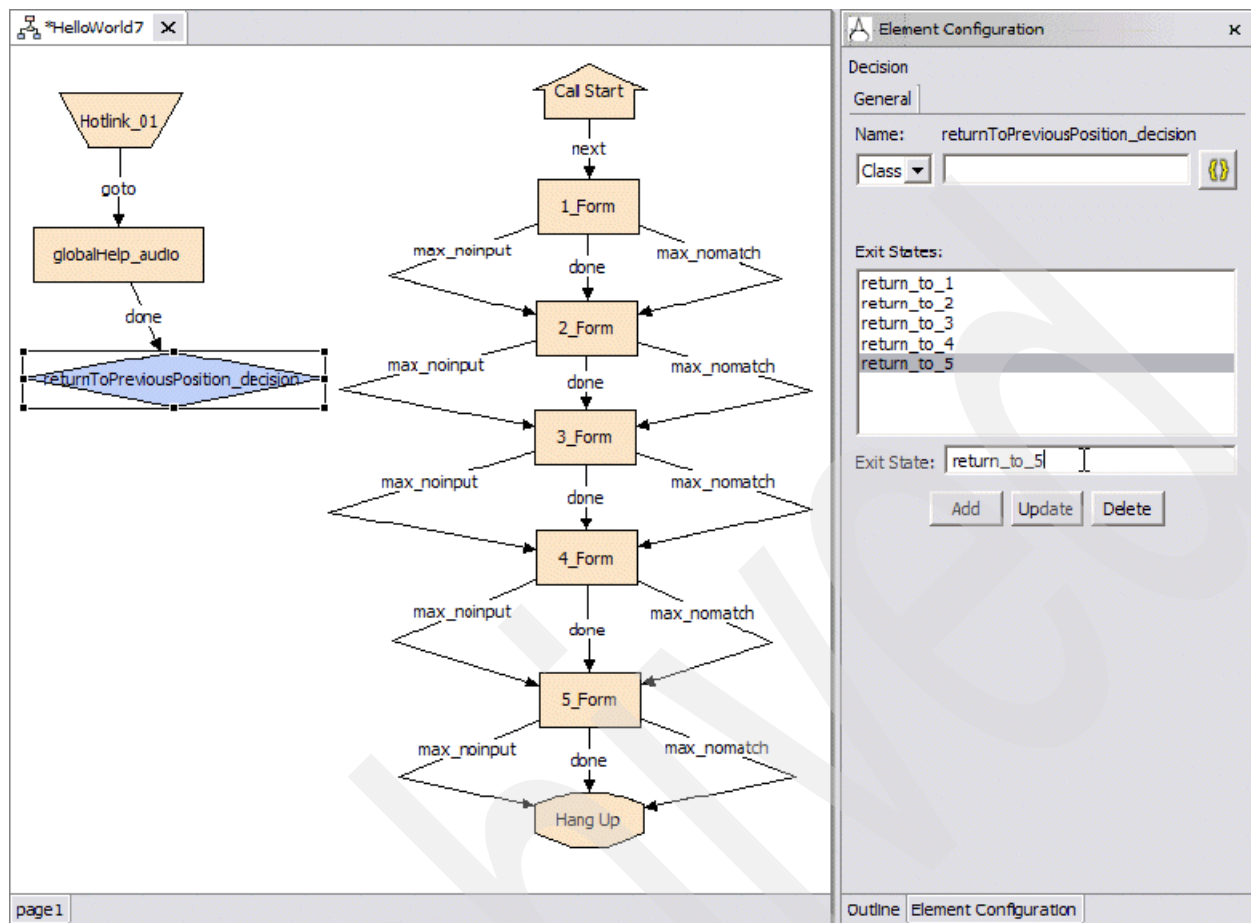


Figure 4-168 HelloWorld7: Adding a decision element

4. Connect the exit states for the decision element as shown in Figure 4-169 on page 242.



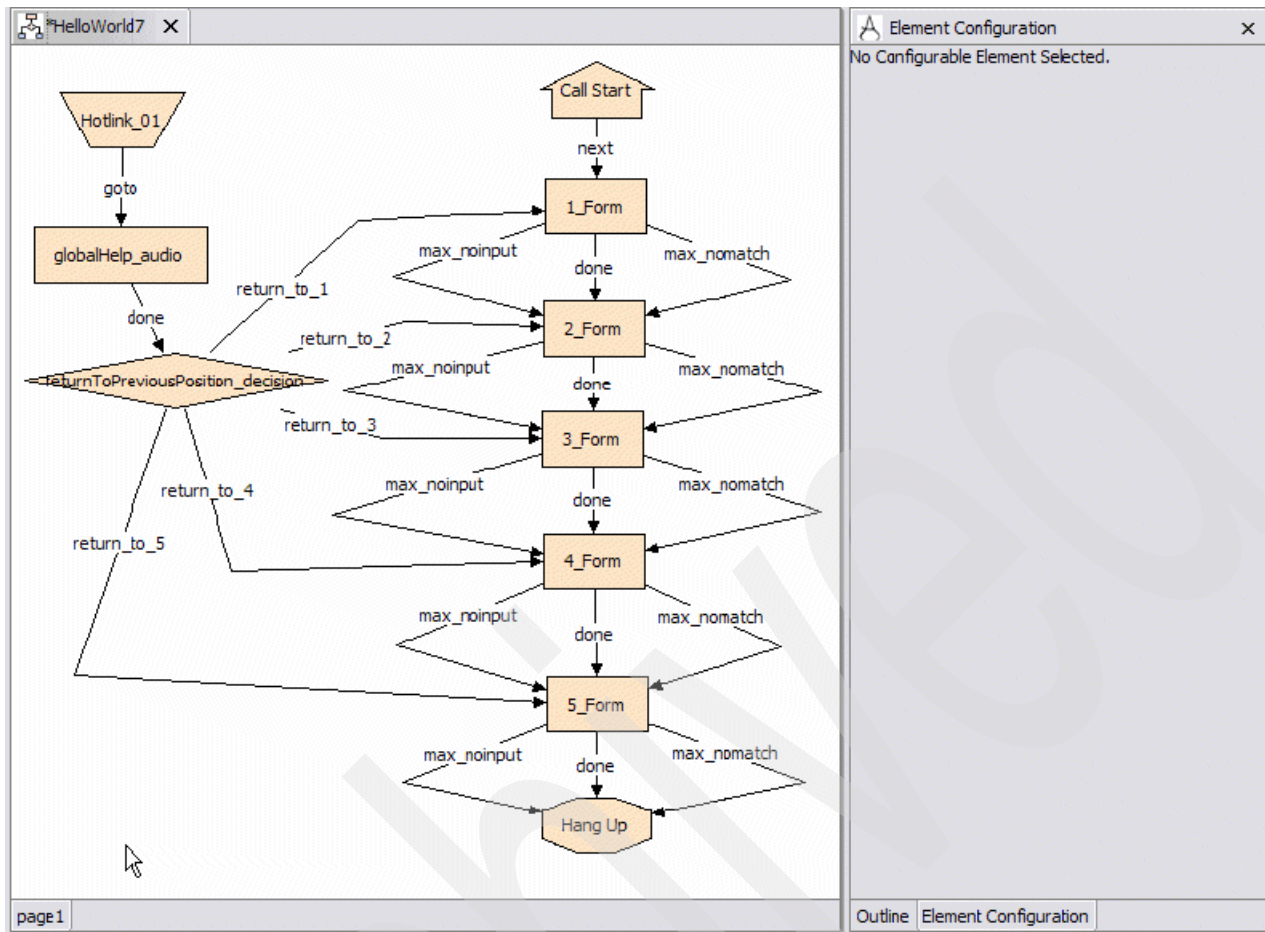


Figure 4-169 HelloWorld7: Connecting the exit states for the decision element

5. At this point we think you can see where we are going with this. The idea is that the `returnToPreviousPosition_decision` will implement logic that will check the value of the `currentPosition` session variable and then exit with the exit state that will throw the user back to that same position in the call flow.

Before we implement that logic, we can use a technique we discussed much earlier. First, we want to ensure that we are correctly capturing the user's position in the flow. Add a TTS string containing a substitution tag to our global help audio that reads out where the user was in the flow when we accessed the global help. Also skip our decision element for now because we have not yet implemented our decision logic. Set up your audio. Add a new audio item that contains the TTS string: The user was at form `{Data.Session.currentPosition}` when this menu was accessed. Please refer to Figure 4-170 on page 243.

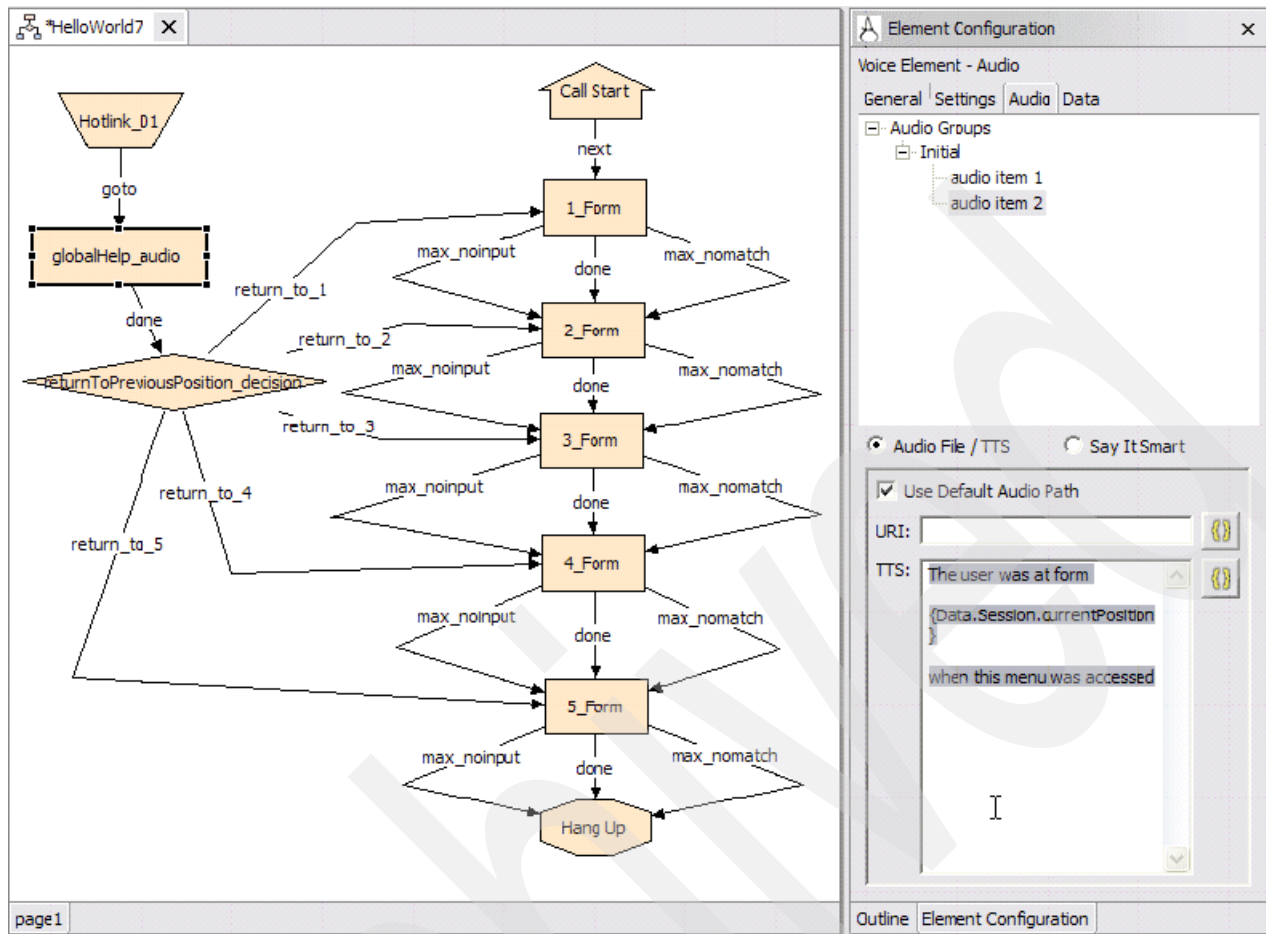


Figure 4-170 HelloWorld7: Adding a TTS string to globalHelp\_audio

6. Skip the decision element so that it always brings us back to the beginning of the call flow as shown in Figure 4-171 on page 244 and Figure 4-172 on page 245.

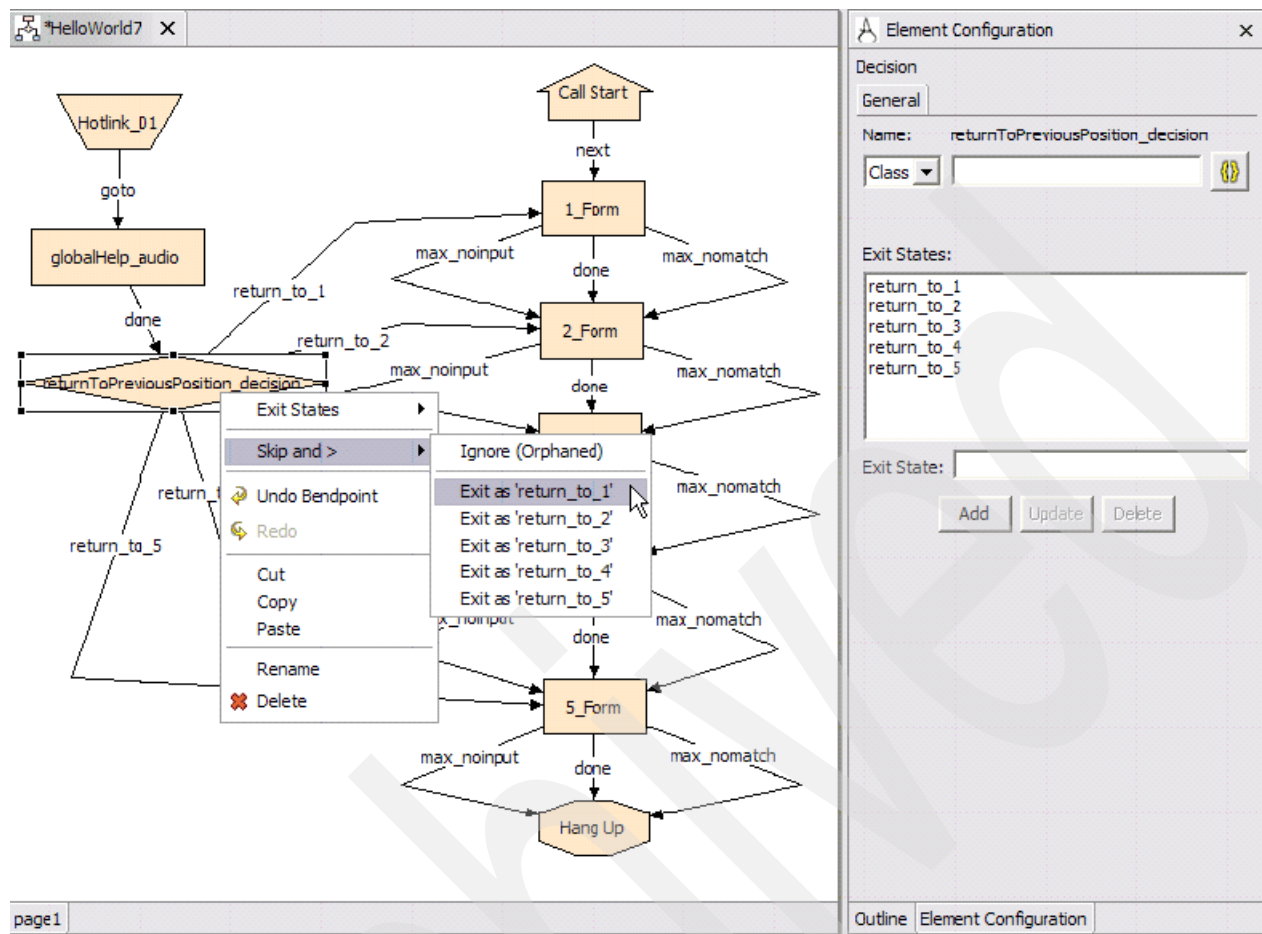


Figure 4-171 HelloWorld7: Skipping the decision element to go back to the beginning of the call flow

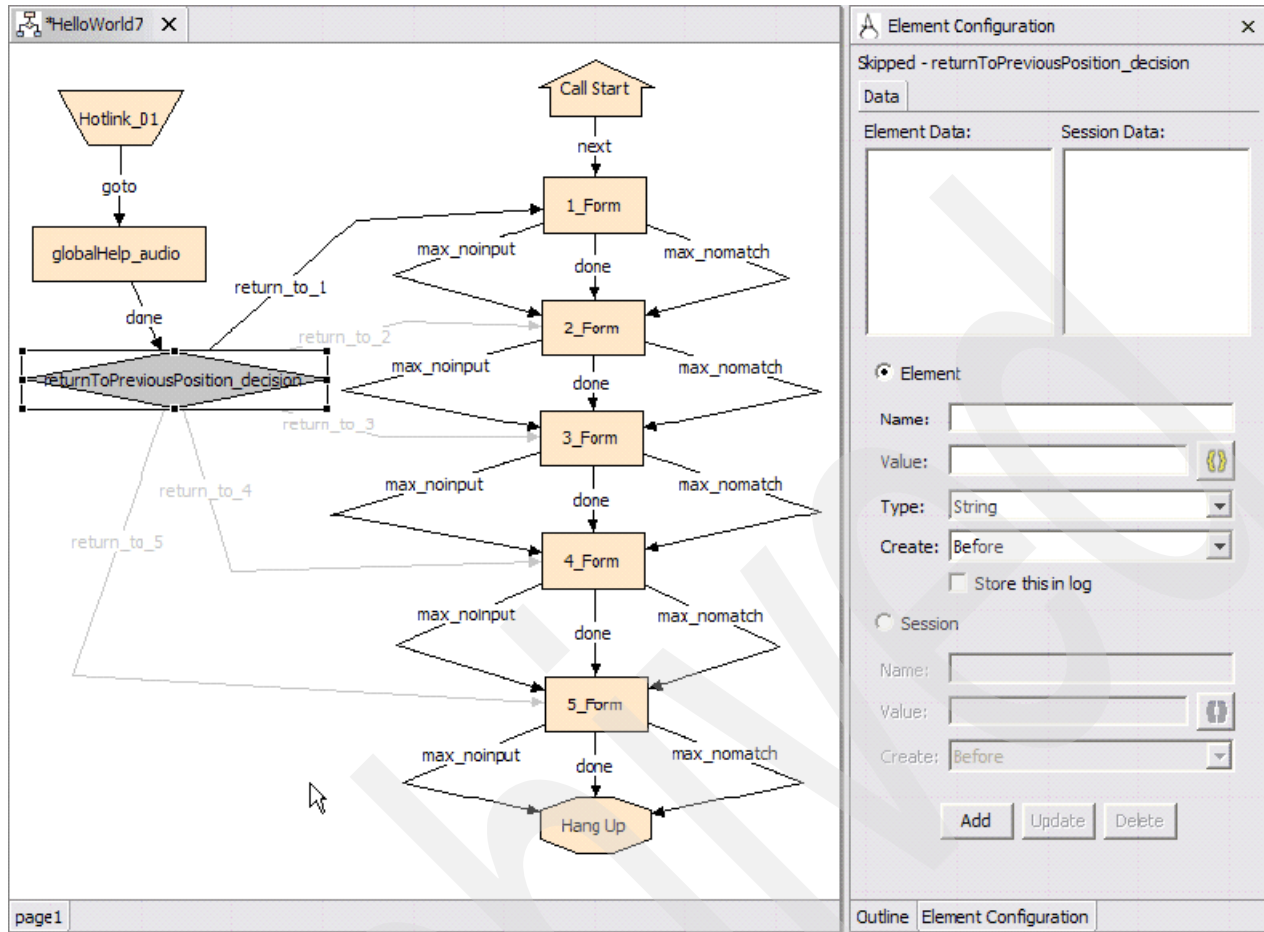


Figure 4-172 HelloWorld7: Result of skipping the decision element

7. Notice how the ability to skip elements gives us the ability to develop in nice manageable increments. Deploy this application and see if it is working as expected and accurately tracking the user's location in the call flow.

We test the new functionality in our application by accessing the global help while at various points in the call flow and we find that our session variable is accurately tracking the callers whereabouts. Now let us look at what we have to do give our decision element the right logic.

### Implementing decision logic with XML API

The beauty of studio is that complex programmatic logic can be implemented without needing to know how to program in Java. The logic for a simple decision can be quite easily implemented using the XML API. We forego a detailed discussion of the XML API here. For much more detailed information about implanting decision logic using the XML API, please refer to the *Cisco CVP VoiceXML V3.1 Programmers Guide* and the *Cisco CVP VoiceXML V3.1 Users Guide* (the section called "CVP VoiceXML XML Decisions in Detail" is a fantastic reference on this topic). We very briefly outline how to create the decision and explain the logic. Because XML is human-readable, the underlying logic should be relatively apparent to the astute observer:

1. *Unskip* our decision element as shown in Figure 4-173 on page 246.

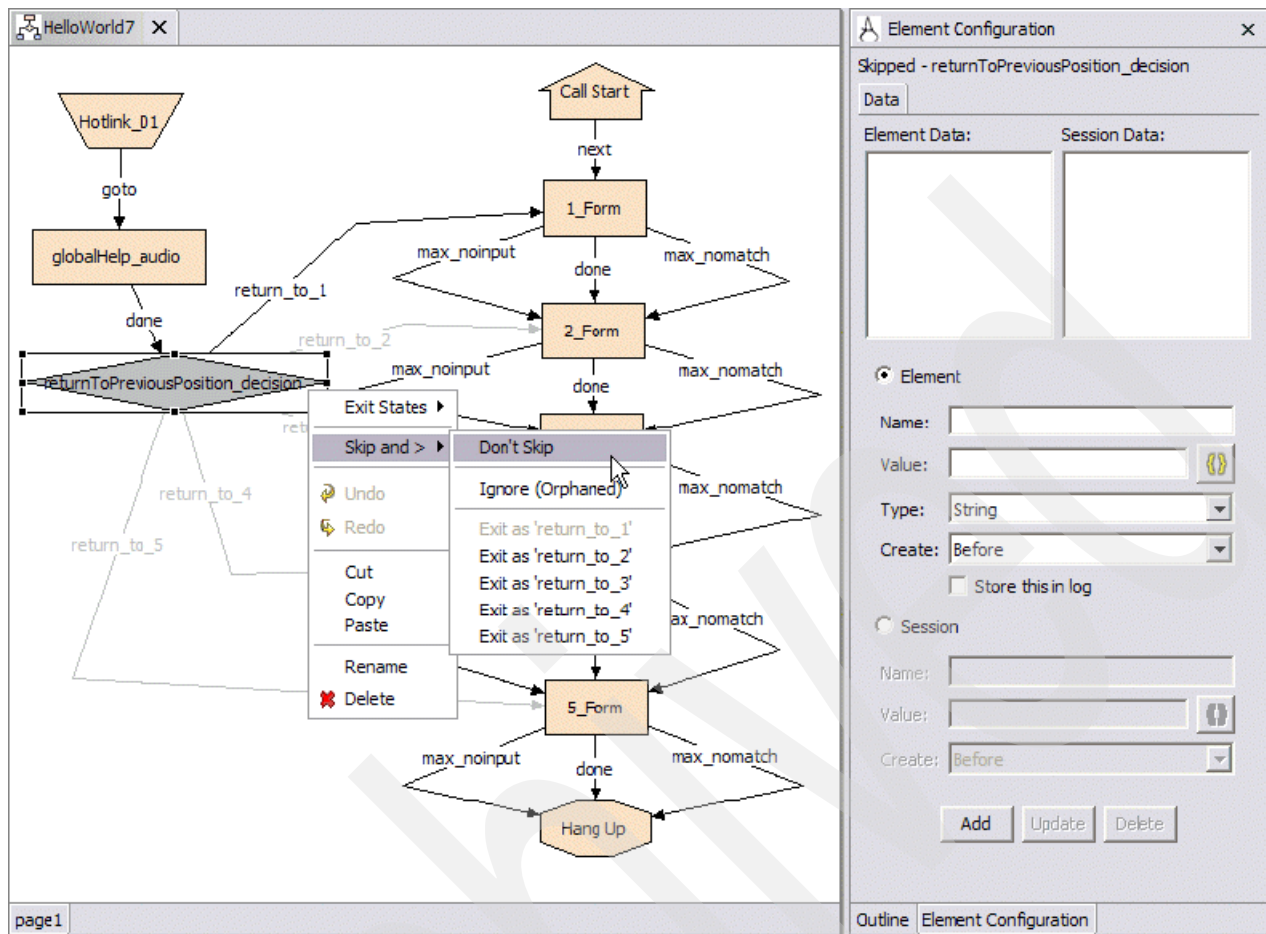


Figure 4-173 HelloWorld7: Unskipping the decision element

2. Configure our decision so that it can use the XML API to implement the underlying logic as shown in Figure 4-174 on page 247.

**Note:** We can also implement decision logic using a java class, which we discuss shortly and also by providing the URI for a page that will return necessary configuration XML to make the decision.



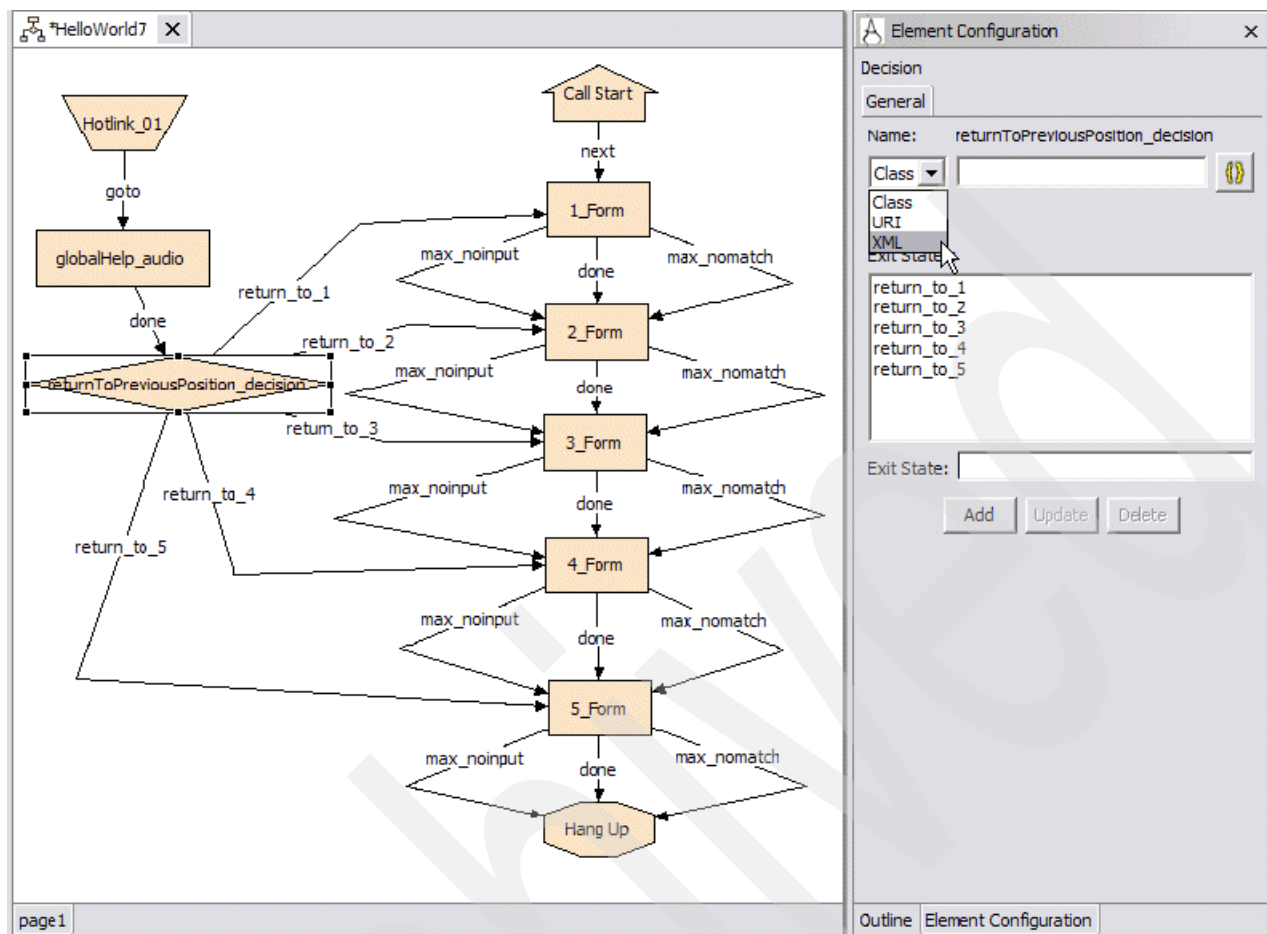


Figure 4-174 HelloWorld7: Configuring the decision element to use the XML API

3. Use the XML Decision Editor (see Figure 4-175 on page 248) to create our decision logic. The Decision Editor can be accessed by clicking the **Edit XML** button.

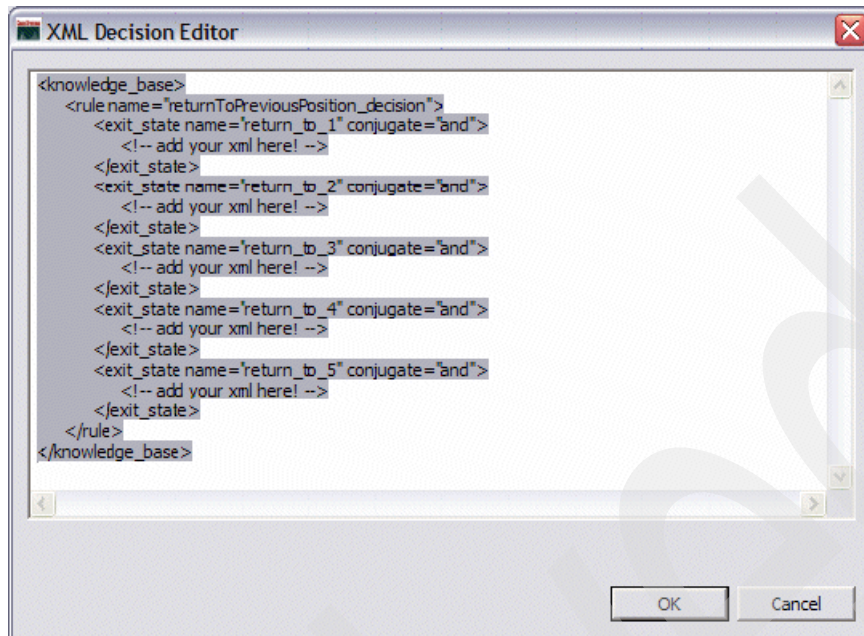


Figure 4-175 XML Decision Editor

As you can see, the XML decision editor appears with the skeleton of our decision logic listing the available exit states. All we need do is provide the conditions that will lead to each one of the exit states and we have successfully implemented our decision logic.

Example 4-6 shows what the final decision XML should look like.

#### Example 4-6 Decision XML listing

```
<knowledge_base>
  <rule name="returnToPreviousPosition_decision">
    <exit_state name="return_to_1" conjugate="and">
      <number operator="equal">
        <data>
          <session name="currentPosition"/>
        </data>
        <constant_number value="1"/>
      </number>
    </exit_state>
    <exit_state name="return_to_2" conjugate="and">
      <number operator="equal">
        <data>
          <session name="currentPosition"/>
        </data>
        <constant_number value="2"/>
      </number>
    </exit_state>
    <exit_state name="return_to_3" conjugate="and">
      <number operator="equal">
        <data>
          <session name="currentPosition"/>
        </data>
        <constant_number value="3"/>
      </number>
    </exit_state>
    <exit_state name="return_to_4" conjugate="and">
```



```

    <number operator="equal">
      <data>
        <session name="currentPosition"/>
      </data>
      <constant_number value="4"/>
    </number>
  </exit_state>
  <exit_state name="return_to_5" conjugate="and">
    <number operator="equal">
      <data>
        <session name="currentPosition"/>
      </data>
      <constant_number value="5"/>
    </number>
  </exit_state>
</rule>
</knowledge_base>

```

---

When this XML code is entered in the XML decision editor, redeploy your application and test it. You should now fall back to the same position you were in when the global help menu was invoked.

### Implementing decision logic with Java API

Decision logic can also be implemented in a compiled Java class. The use of Java in implementing voice application functionality is far more versatile, robust and full-featured than the XML API. The only drawback is some Java programming knowledge is naturally required. We forego a thorough discussion of CVP-oriented Java programming here. If you require more detailed information about this topic, please refer to the *Cisco CVP VoiceXML V3.1 Programmers Guide* and the Java docs for the CVP API. We modify our application so that it uses a Java class called `PositionDecisionClass` as the source of its decision making logic, as shown in Figure 4-176 on page 250.

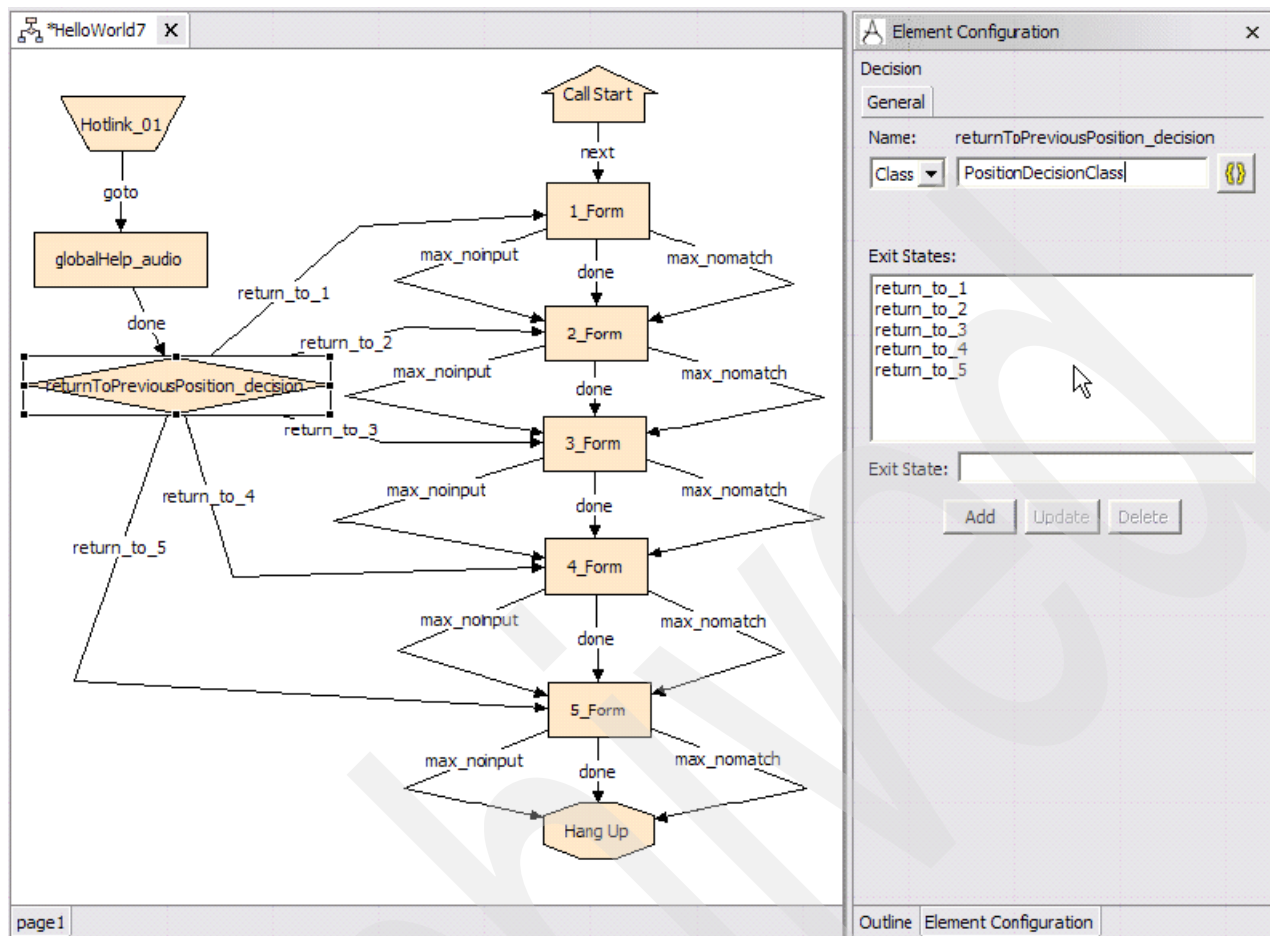


Figure 4-176 HelloWorld7: Creating a Java class for the decision element

Now, you can create the Java class in any environment you desire. It is very easy to simply create it from within Studio. Studio after all is nothing more than an Eclipse plug-in and Eclipse is one of the world's foremost Java development platforms. Just create a new Java project as shown in Figure 4-177, Figure 4-178 on page 251, Figure 4-179 on page 251, and Figure 4-180 on page 252.

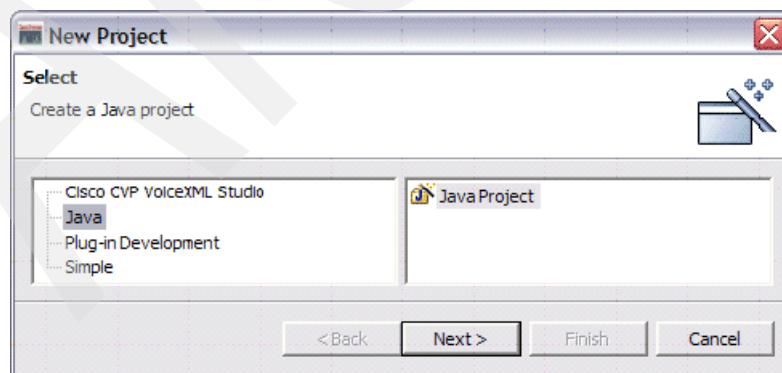


Figure 4-177 Create a Java project

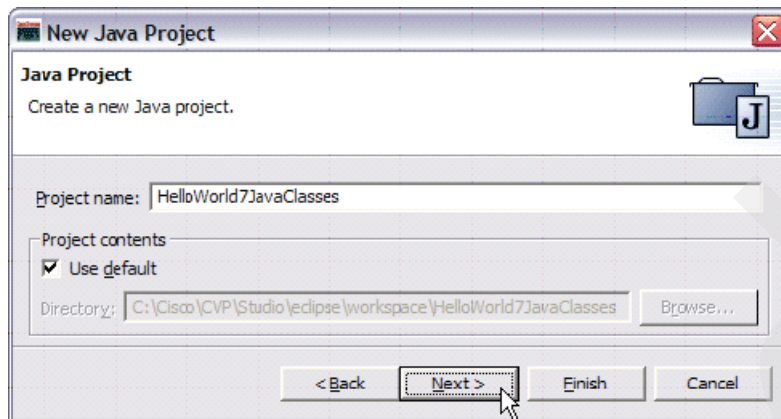


Figure 4-178 New Java Project: Specifying the project name

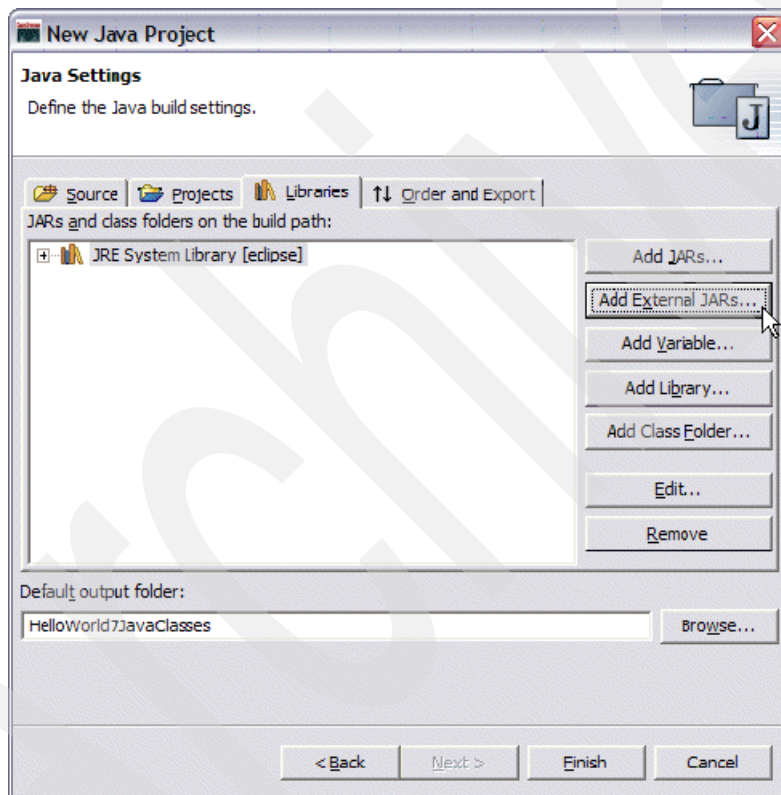


Figure 4-179 New Java Project: Add External JARS...

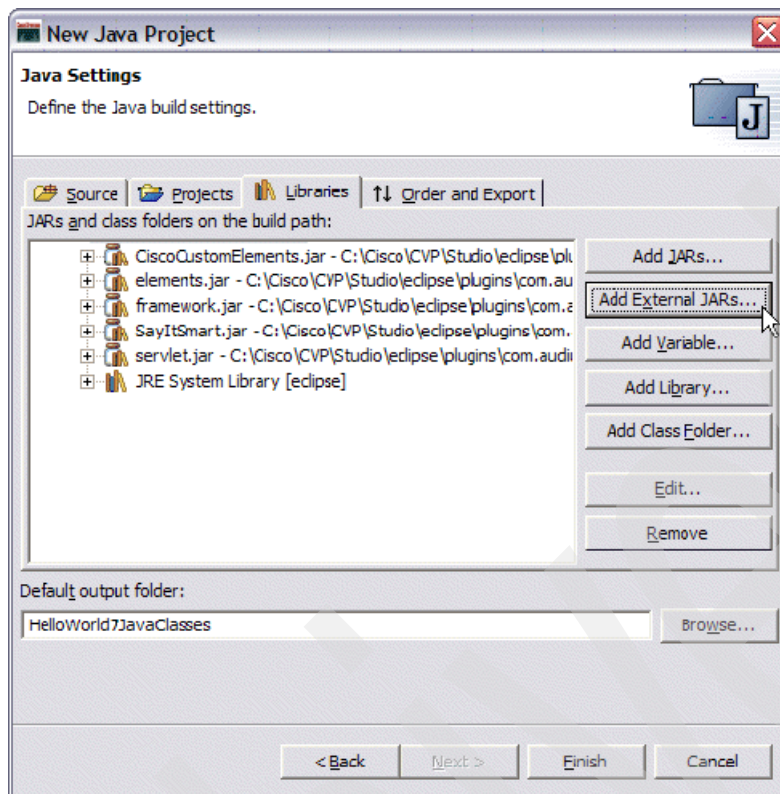


Figure 4-180 New Java Project: Result of adding external JARS

**Note:** Generally, for all CVP Java projects, you must include the following jar files in your build path: elements.jar, framework.jar, SayItSmart.jar and servlet.jar.

Create a class called PositionDecisionClass and enter the code shown in Example 4-7.

*Example 4-7 PositionDecisionClass.java*

```
import com.audium.server.AudiumException;
import com.audium.server.voiceElement.*;
import com.audium.server.session.DecisionElementData;
public class PositionDecisionClass extends DecisionElementBase
{
    public String doDecision(String name, DecisionElementData decisionData) throws
    AudiumException
    {
        String exitState = "";
        if(decisionData.getSessionData("currentPosition").toString().equals("1"))
        {
            exitState = "return_to_1";
        }
        if(decisionData.getSessionData("currentPosition").toString().equals("2"))
        {
            exitState = "return_to_2";
        }
        if(decisionData.getSessionData("currentPosition").toString().equals("3"))
        {
            exitState = "return_to_3";
        }
    }
}
```

```

        if(decisionData.getSessionData("currentPosition").toString().equals("4"))
        {
            exitState = "return_to_4";
        }
        if(decisionData.getSessionData("currentPosition").toString().equals("5"))
        {
            exitState = "return_to_5";
        }
        return exitState;
    }
}

```

---

Of course a much more succinct and elegant way (and slightly more cryptic) to accomplish this would be to implement the doDecision method in the following way as shown in Example 4-8.

*Example 4-8 Alternate doDecision method*

---

```

public String doDecision(String name, DecisionElementData decisionData) throws
AudiumException
{
    String exitState = "";
    try
    {
        int num = new
Integer(decisionData.getSessionData("currentPosition").toString()).intValue();
        return "return_to_" + num;
    }
    catch(Exception e)
    {
        return "return_to_1";
    }
}

```

---

Now build the project as shown in Figure 4-181 on page 254.

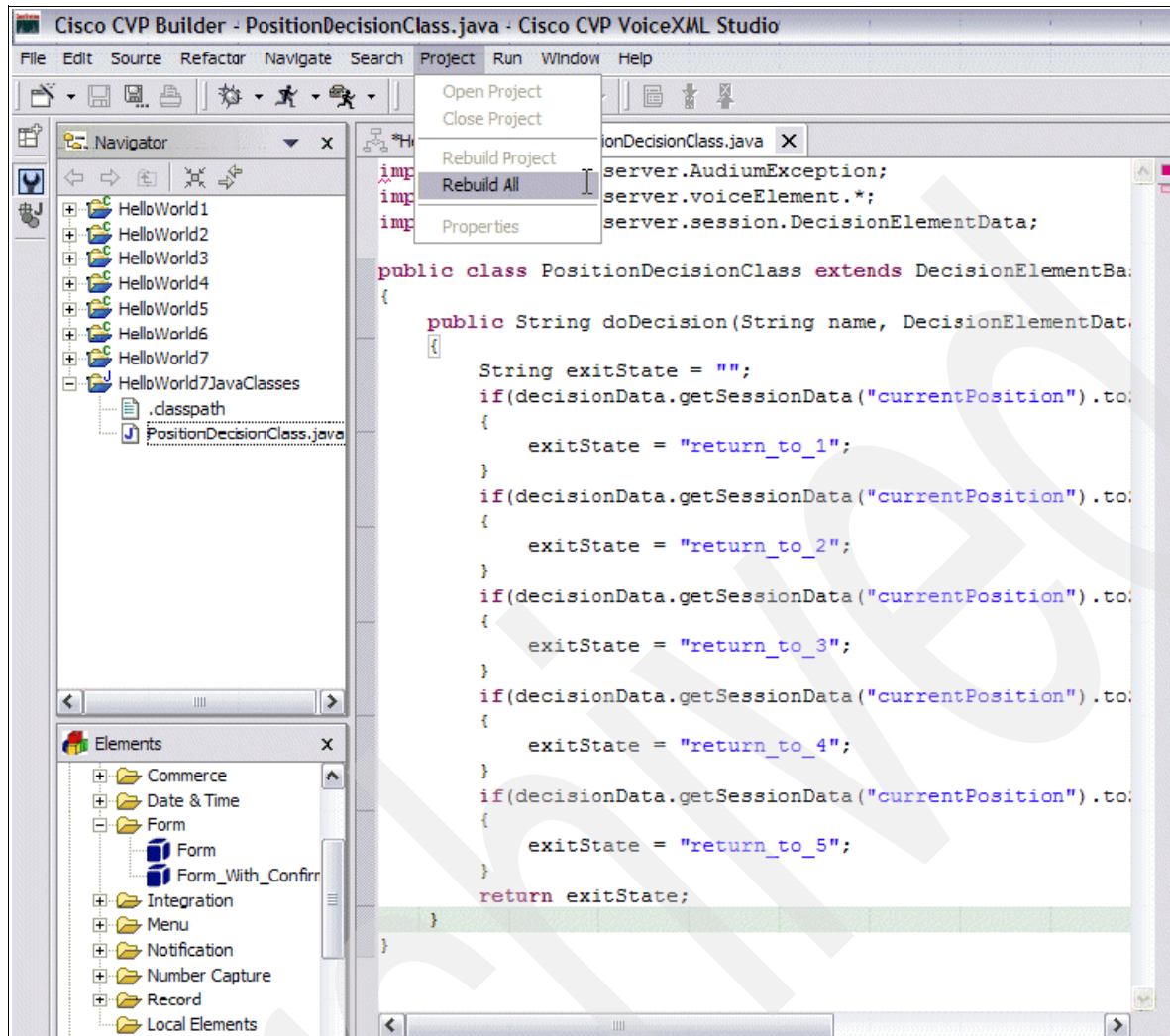


Figure 4-181 Rebuild All

Copy and paste the compiled class to the deploy/java/application/classes folder of the HelloWorld7 project (see Figure 4-182 on page 255, through Figure 4-184 on page 256). This is where all classes that will be used in the voice application should be placed.



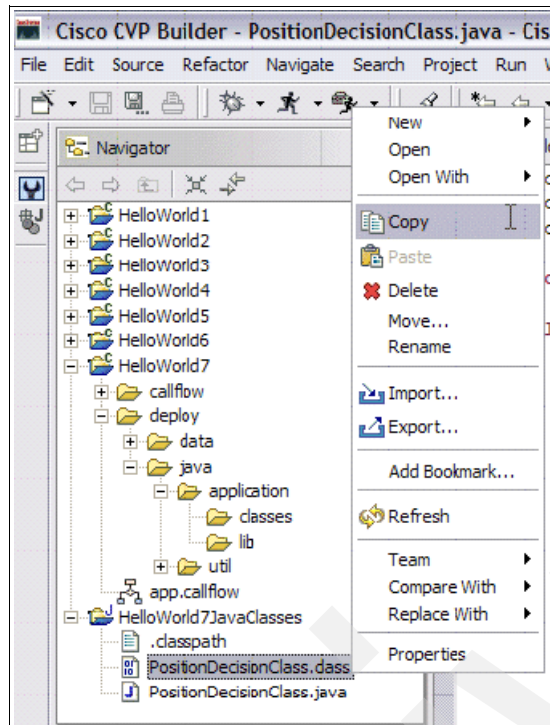


Figure 4-182 Copy PositionDecisionClass.class

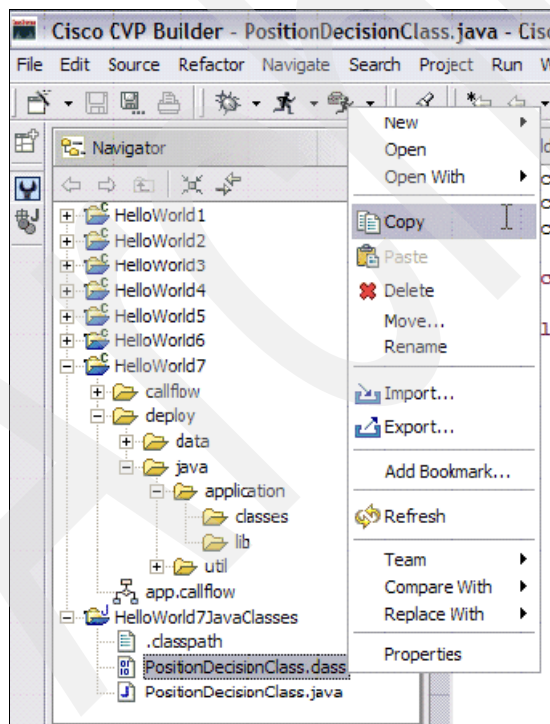


Figure 4-183 Paste PositionDecisionClass.class into the deploy/java/application/classes folder



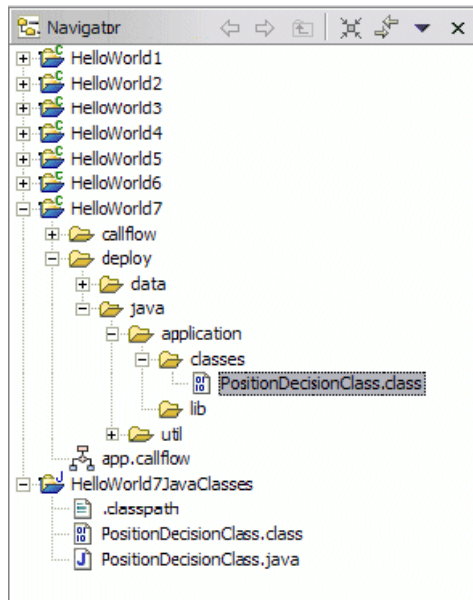


Figure 4-184 Result of copying *PositionDecisionClass.class* into *deploy/java/application/classes*

You can now deploy your application and you will see that the functionality will be exactly the same as that you experienced when implementing the decision logic with the XML API.

### HelloWorld9: add global error handling

We are now at a point where we have covered a very large number of introductory concepts which should, in theory, have provided you with a fairly extensive bag of tricks that you can call on when developing your own applications. For the final two examples, we look at some more advanced concepts that you might need to employ.

The first of these is the introduction of global error handling for your application. There is more than one way to approach global error handling. One way is to define hotevents that will catch errors that we might anticipate while in the design phase. For example, we know that a missing audio or grammar file might result in the raising of an `error.badfetch` event, which could be easily caught and processed by a configurable hotevent element. However, this approach requires that we anticipate all possible exceptional conditions in order to provide certainty of a graceful exit from the application, regardless of what might have triggered the crash.

The other approach, which is somewhat more robust, would be to configure a voice element so that it acts as an error element. An *error element* is defined as: “A voice element whose purpose is to handle the caller when an unexpected error occurs.” The beauty of this is that any configurable voice element can be turned into an error element simply by right clicking the element in the call flow pane and selecting **Error Element** → **Yes**. Once this is done, the element (which can be any configurable voice element) now has the capability of catching all unhandled exceptions. As you can imagine, the fact that any voice element can be transformed into an error element allows for a great deal of flexibility when dealing with unhandled exceptions. Furthermore, error elements used in conjunction with hotevent elements (hotevents provide predefined processing for anticipated exceptional conditions and error element catches any unanticipated errors) allow the application designer to have complete and robust error recovery mechanisms built into all of their applications at design time. For HelloWorld9, we tweak one of our earlier applications (HelloWorld4 –the application with the form and external grammar) which contains a form with an external grammar. It is a well known fact to voice developers (alas it will become well known to you as well if you are

relatively new to this) that missing external grammars result in the raising of an error.badfetch event. So, if we run our HelloWorld4 application and the external grammar file is not in the location that the application expects it to be, then we can also expect that the application will crash and the user will be rather rudely disconnected from the application with the generic error message. Try it out. If we wish to provide more graceful handling, we must introduce an error element to the call flow. First, create a copy of the HelloWorld4 project and call it HelloWorld9.

1. Now we know that the application will look for the external grammar file names.grxml at:

`http://9.42.171.24:8080/CVP/grammars/HelloWorld4/names.grxml`

Remember this is the location specified for our lab setup. Your path will very likely have a different IP address, but if you structured your directories in the way specified in HelloWorld4 the URL should otherwise be the same. We modify this so that it looks for the grammar file in the CVP/HelloWorld9 directory, i.e. change the path to:

`http://9.42.171.24:8080/CVP/grammars/HelloWorld9/names.grxml`

2. Create an empty directory on your CVP app server machine:

`C:/cisco/cvp/tomcat 4.1/webapps/cvp/grammars/helloworld9`

3. Now, the form will expect to find the names.grxml file there. But, of course, it will never find it. Validate, deploy, and try running your application. It should crash out with the generic error message as expected. Now, let us add an error element that will handle this condition to our call flow:
4. Add a new audio element to the call flow and change it to an error element. Remember, an application can only have one error element in it. Please refer to Figure 4-185.

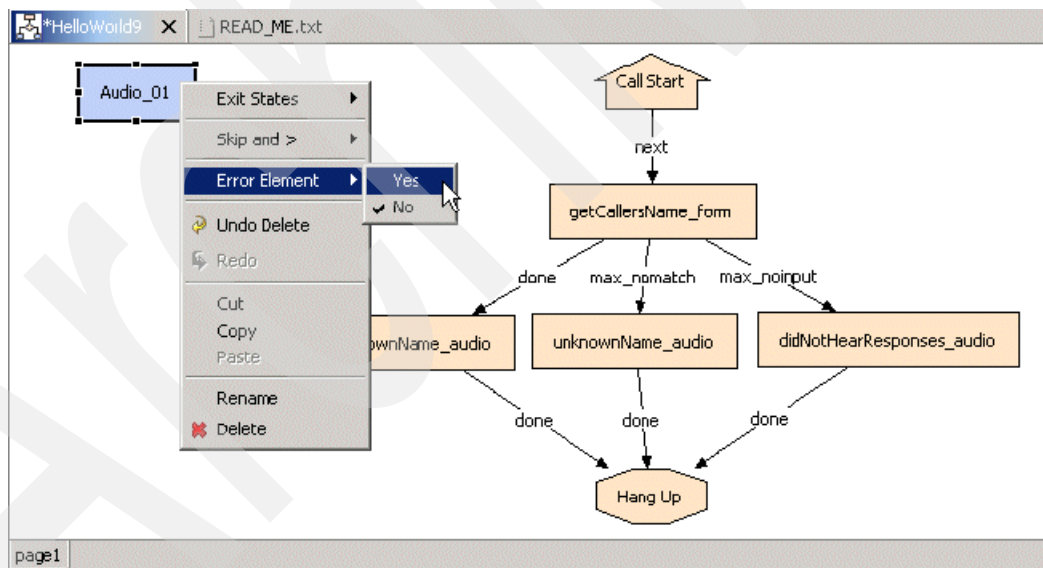


Figure 4-185 Changing the audio element into an error element

Give it a name consistent with our standard nomenclature (see Figure 4-186 on page 258).

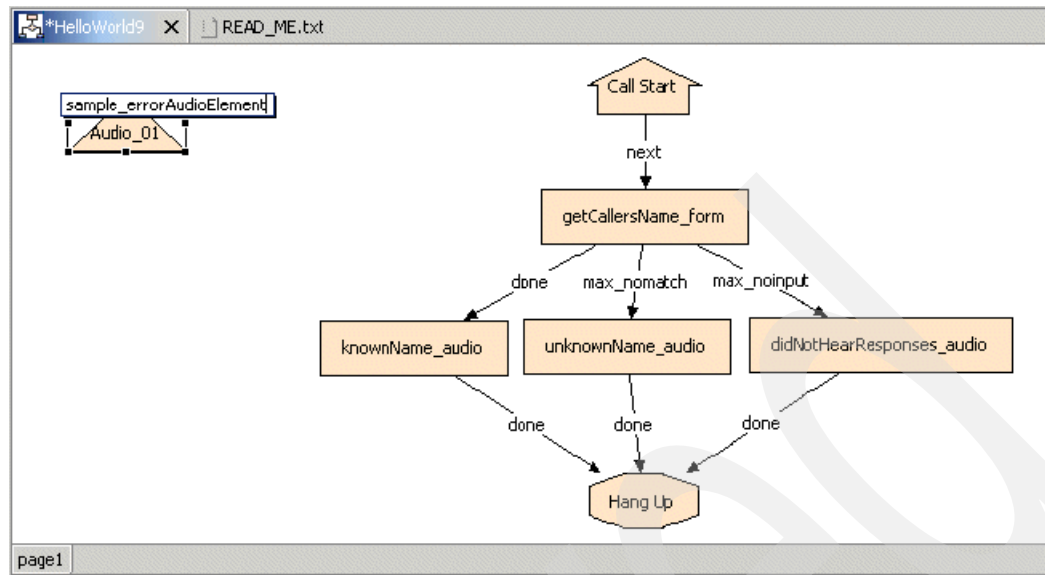


Figure 4-186 Renaming the element to sample\_errorAudioElement

- Define some TTS content for the initial audio group as shown in Figure 4-187.

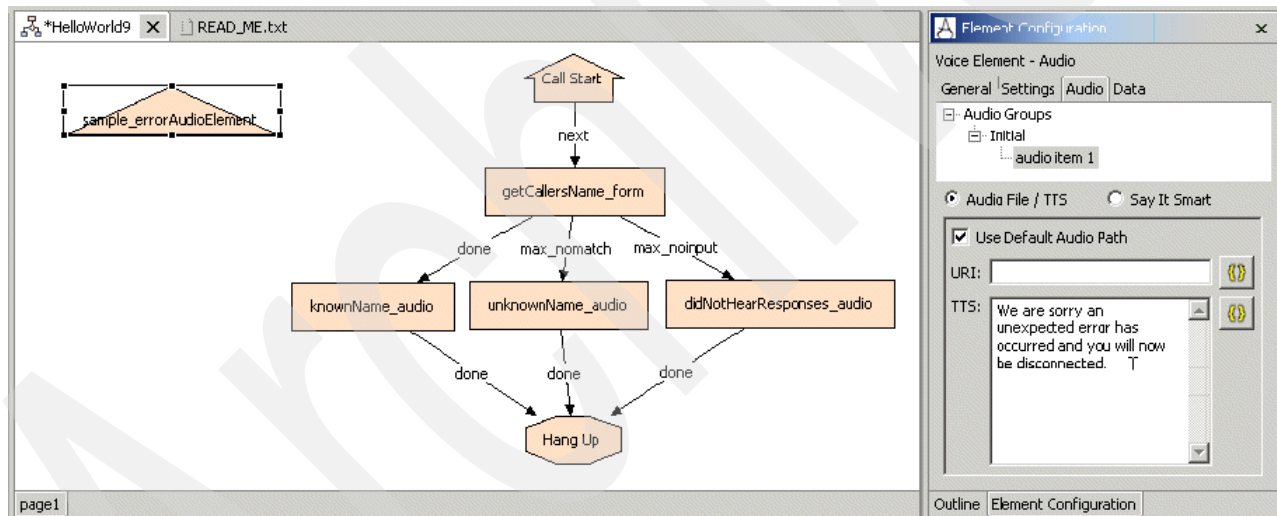


Figure 4-187 Defining TTS content for the initial audio group

- We can now validate and deploy our application. You will notice that when the application is run that the message We are sorry an unexpected error has occurred and you will now be disconnected is played, rather than the standard generic error message you were hearing before we added the error element to the application.

**Design Tip:** When creating error elements, it is usually a good idea to provide both prerecorded audio and TTS content for all audio groups. This is because unexpected errors can sometimes arise as a result of a TTS engine failure. If the TTS engine is for some reason unavailable upon invocation of error handling code, then the audio files should relay the needed information to the user.

Now, that was clearly not a very advanced or tightly engineered example of error handling in a voice application, but it does illustrate the use of an error element in a CVP application. We

leave it to you to design and implement more robust error handling schemes in your own work.

## Application debugging, important log files and their interpretation

Without a doubt one of the most powerful and helpful resources available to the Cisco CVP application developer when debugging and troubleshooting their applications is the log files created by the CVP application server. The log files are a rich source of information and can be used to isolate the source of the vast majority of runtime errors. Validation errors are usually very easy to fix because they can arise for a fairly limited number of reasons. Runtime errors in voice applications, however, can be extremely difficult to pin down. It is not uncommon to waste many hours diagnosing and remedying very simple runtime errors until a certain level of familiarity with the analysis of various log files is obtained by the developer. We first examine the purposes of various individual log files and then run through some examples of analyzing those log files in order to diagnose and fix runtime errors.

### Call log

The call log records a single line for every application visit handled by the Cisco CVP server. For the most part, because a single call entails only a visit to a single application, each line in the call log is equivalent to a single physical call placed into the system. Under a default CVP server install, the server call logs can be found in the following location:

```
C:\Cisco\CVP\Server\logs
```

Each line in the call log contains the following fields:

- ▶ CallID
- ▶ SessionID
- ▶ Callers
- ▶ Order
- ▶ Application
- ▶ Time

The meaning of these fields can be found in Example 4-9 along with some sample entries from a real-life call log. Example 4-9 shows a sample call log from a Cisco CVP server running two applications. The sample call log is in the following comma delimited format:

```
CallID,SessionID,callers,Order,Application,timestamp
```

#### Example 4-9 Sample call log from a running CVP server

---

```
128.1.1.246.1110203142455.0,128.1.1.246.1110203142455.0.PackageTracking,0,1,PackageTracking,03/07/2005
08:45:42
128.1.1.246.1126097382616.1,128.1.1.246.1126097382616.1.PackageTracking,0,1,PackageTracking,09/07/2005
08:49:42
128.1.1.246.1126097718561.2,128.1.1.246.1126097718561.2.PackageTracking,0,1,PackageTracking,09/07/2005
08:55:18
128.1.1.246.1126098890033.3,128.1.1.246.1126098890033.3.CreditCard,0,1,CreditCard,09/07/2005 09:14:50
128.1.1.246.1126099877639.4,128.1.1.246.1126099877639.4.PackageTracking,0,1,PackageTracking,09/07/2005
09:31:17
128.1.1.246.1126100007621.5,128.1.1.246.1126100007621.5.PackageTracking,0,1,PackageTracking,09/07/2005
09:33:27
128.1.1.246.1126102423700.6,128.1.1.246.1126102423700.6.CreditCard,0,1,CreditCard,09/07/2005 10:13:43
128.1.1.246.1126102633838.7,128.1.1.246.1126102633838.7.PackageTracking,0,1,PackageTracking,09/07/2005
10:17:13
```

---

Table 4-10 explains the various call log fields.

Table 4-10 Call log fields and their meaning

Field Name	Meaning
CallID	This is a non-repeating value generated by the Server to uniquely identify calls. It is designed to be unique even across machines, as the log files of multiple machines running the same applications may be combined for analyses. The format of the session ID is IP.SECS.INCR where IP is the IP address of the machine on which the Server is installed, SECS is a large integer number representing the time the application visit was made and INCR is an automatically incremented number managed by the Server. Each part is delimited by dots and contains no spaces. For example, 192.168.1.121.1024346999979.1.
SessionID	The session ID (generated by the CVP server) is used to track a visit to a specific application. Note: with application transfers, one call ID may be associated with multiple session IDs. For this reason, session IDs are simply the call ID with the application name appended to the end. For example, 192.168.1.121.1024346999979.1.TestApp.
callers	An integer representing the total number of callers interacting with the system at the time the call was received (includes the current call in the tally).
order	A number indicating the order of each application visited in a call. The order begins at 1. This column exists to report the order in which a caller visited each application should the data be imported to a database.
Application	The name of the application visited.
time	A timestamp of the application visit in a readable format. This represents when the call was received or the application transfer occurred.

As a troubleshooting tool, the call log is perhaps a little less useful than some of the other logs generated by the server. It is however a very useful historical reporting and auditing tool.

### Application activity log

Without a question or a doubt the application activity log is one of the most important troubleshooting tools available to the voice application developer. You can think of the activity log as essentially a listing of all steps and turns taken by individual callers whenever they visit an application deployed to the Cisco CVP server. Why is this log so important? In many cases, it enables you to pinpoint exactly where in the application something went wrong by showing exactly how far the caller progressed before an exceptional condition was triggered. The application activity logs are located in the individual application's log directory. For example, for the application HelloWorld9 deployed on a default install of the Cisco CVP server these logs will be found in:

```
C:/cisco/cvp/server/applications/HelloWorld9/logs
```

There are quite a few fields present for each entry in the call activity log shown in Example 4-10. Table 4-11 containing explanations of various fields and values you can encounter while perusing an activity log as well as a sample from a real-life application log:

#### Example 4-10 Call activity log

```
128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005 13:50:12,,start,newcall,
128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005 13:50:12,,start,ani,NA
128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005 13:50:12,,start,areacode,NA
128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005 13:50:12,,start,exchange,NA
128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005 13:50:12,,start,dnis,NA
128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005 13:50:12,,start,iidigits,NA
128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005 13:50:12,,start,uui,NA
```

```

128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005
13:50:12,getCallersName_form,element,enter,
128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005 14:20:28,,end,how,app_session_complete
128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005 14:20:28,,end,result,timeout
128.1.1.246.1132253412160.0.HelloWorld9,11/17/2005 14:20:28,,end,duration,1815

```

*Table 4-11 Call activity log fields and their meaning*

Field Name	Meaning
SessionID	The session ID (generated by the CVP server) is used to track a visit to a specific application. Note: with application transfers, one call ID may be associated with multiple session IDs. For this reason, session IDs are simply the call ID with the application name appended to the end. For example: 192.168.1.121.1024346999979.1.TestApp.
[ElementName]	The name of the current element the activity belongs to. Only functional elements (voice elements, action elements, decision elements, and insert elements) can appear here. This column would be empty if the activity does not apply to an element.
Category	The category of the action (see Table 4-12 on page 261 for a more complete description of various actions and categories)
Action	A keyword indicating the action taken.
Description	Some qualifier or description of the action.
Tine	Timestamp

Table 4-12 on page 261 shows the various category and action descriptions that appear in the call activity log.

*Table 4-12 Category and action descriptions that appear in the call activity log*

Category	Action	Description
start	newcall or source	newcall is used when the application visit is a new call. The description is empty. source is used when another application transferred to this application. The name of the application transferred from is listed in the description.
start	ani	The description is the ANI of the caller. NA if the ANI is not sent.
start	areacode	The area code of the ANI. NA if the ANI is not sent.
start	exchange	The exchange of the ANI. NA if the ANI is not sent.
start	dnis	The description is the DNIS of the call. NA if the DNIS is not sent.
start	iidigits	The description is the IIDIGITS of the call. NA if the IIDIGITS is not sent.
start	uui	The description is the UUI of the call. NA if the UUI is not sent.
start	uid	The application visit is associated with a user. The UID is listed in the description.

Category	Action	Description
start	onhold	The caller was placed on hold before the call began because there were no available Cisco CVP VoiceXML ports. The description contains how many times the on hold message was played to the caller.
start	error	An error occurred in the on call start action (either a Java class or XML-over-HTTP). The description is the error message.
end	how	How the call ended. The description is either hangup to indicate the caller hung up, disconnect to indicate the system hung up on the caller, application_transfer to indicate a transfer to another CVP VoiceXML application occurred, call_transfer to indicate a telephony blind transfer occurred, or app_session_complete to indicate that the call session ended via another means such as a timeout or the call being sent to an IVR system outside of Cisco CVP VoiceXML.
end	result	The description explains why the call ended. normal indicates the call ended normally, suspended indicates the application is suspended, max_ports indicates the caller hung up while on hold before the application started, error indicates an error occurred, timeout indicates that the Server session timed out, and invalidated indicates the application itself invalidated the session.
end	duration	The duration of the call, in seconds.
end	error	An error occurred in the on call end action (either a Java class or XML-over-HTTP). The description is the error message.
element	enter	The element was entered. The description is empty. This is always the first action for an element.
element	hotlink	A hotlink was activated while in the element. The description lists the hotlink name.
element	hotevent	A hotevent was activated while in the element. The description lists the hotevent name.
element	error	An error occurred while in the element. The description lists the error message.
element	flag	A flag was triggered. The description lists the flag name.
element	exit	The element was exited. The description lists the exit state of the element or is empty if a hotlink, hotevent or error occurred within the element.
interaction	audio_group	An audio group was played to the caller. The description is the audio group name.
interaction	inputmode	How the caller entered data. The description can be dtmf or speech.



Category	Action	Description
interaction	utterance	The caller said something that was matched by the speech recognition engine. The description lists the match it made of the utterance. This action will always appear with the interpretation and confidence actions.
interaction	interpretation	In a grammar, each utterance is mapped to a certain interpretation value. The description holds the interpretation value for the caller's utterance. This action will always appear with the utterance and confidence actions.
interaction	confidence	The confidence of the caller's matched utterance. This is a decimal value from 0.0 to 1.0. DTMF entries will always have a confidence of 1.0. This action will always appear with the utterance and interpretation actions.
interaction	nomatch	The caller said something that did not match anything in the grammar.
interaction	noinput	The caller did not say anything after a certain time period.
data	[NAME]	When an element creates element data, one can specify if to log the element data. Element data slated to be logged will appear here with the element data name as the action and the value as the description.
custom	[NAME]	Anywhere the developer adds custom name/value information to the log will have the name appear as the action and the value stored within as the description.

The amount of information contained in the application activity logs can be tweaked by modifying the logging level property in your applications properties (Table 4-13 shows a listing of what information is included for various levels of logging verbosity).

Table 4-13 Logging verbosity levels

Category	Description	Minimal	Moderate	Complete
Start	Information on new visits to the application. The element column is empty for this category.	X	X	X
End	Information on how the application visit ended. The element column is empty for this category.	X	X	X
Element	Information on the elements visited and how the element was exited.		X	X

Category	Description	Minimal	Moderate	Complete
Interaction	Detailed information about what a caller did within a voice element.			X
Data	Element data to be logged			X
custom	Custom developer-specified data to log.			X

### **Application error log**

The application error logs are another essential source of information for troubleshooting and debugging purposes. The call error logs are essentially a distilled version of the application activity logs that include only information about errors. A java stack trace should be included for any classes that raise application errors. The fields shown in Table 4-14 on page 264 can be found in the application error logs:

*Table 4-14 Application error log fields and their meanings*

Field Name	Meaning
SessionID	The session ID (generated by the CVP server) is used to track a visit to a specific application. Note: with application transfers, one call ID may be associated with multiple session IDs. For this reason, session IDs are simply the call ID with the application name appended to the end. For example: 192.168.1.121.1024346999979.1.TestApp.
Description	The error description including Java stack trace if applicable.
Time	Timestamp

### **Call error log**

There are rare occasions where errors might occur on the Cisco CVP server outside of the context of an individual application. Example 4-11 illustrates one of these types of errors. One common example of this is a licensing error (max\_ports), which occurs when too many concurrent calls are received by the server resulting in consumption of all available licenses.

*Example 4-11 Sample entries from various call error logs*

```
128.1.1.246.1126102633838.7.PackageTracking,09/07/2005 10:19:10, SERVER ERROR: A VoiceXML
exception occurred: error.com.cisco.media.resource.failure.asr
128.1.1.246.1110203142455.0.PackageTracking,03/07/2005 08:47:17, SERVER ERROR: An element
encountered an exception of type com.audium.server.xml.ElementConfigException with the
message: The Say It Smart plugin "com.audium.server.xml.c@a8e586" has no filesets defined
for the output format "com.audium.server.xml.c@74cb02".
```

The fields shown in Table 4-15 are present in the call error logs.

Table 4-15 Call error log fields and their meanings

Field Name	Meaning
Description	The error description. One possible value can be <i>max_ports</i> , indicating the caller was put on hold because all the CVP VoiceXML license ports were taken. While the call was eventually handled correctly, this is placed here as a notice that he license may not have enough CVP VoiceXML ports to satisfy caller demand. Another value is <i>bad_url</i> : <i>[URL]</i> , indicating that a request was made to Server for a URL that could not be recognized. This most likely will occur if the voice browser refers to an application that either does not exist or no longer exists. The last description is <i>error</i> , indicating that some other error occurred.
Time	Timestamp

### Server and application administrative history log

Whenever one of the administrative scripts is run (either against an individual application or the server as a whole), an entry is created in the application and server admin history logs.

The events shown in Table 4-16 can be seen in the admin history logs:

Table 4-16 Descriptions of events seen in admin history logs

Event	Description
Server_start	Listed when the Java application server starts up. Each application's history log contains records of each time the application server starts.
Deploy_app	Listed when the <b>deployApp</b> script is run.
Suspend_app	Listed when the <b>suspendApp</b> script is run.
Resume_app	Listed when the <b>resumeApp</b> script is run.
Update_app	Listed when the <b>updateApp</b> script is run.
Deploy_all_new_apps	Listed when the <b>deployAllNewApps</b> script is run.
Flush_all_old_apps	Listed when the <b>flushAllOldApps</b> script is run.
Suspend_server	Listed when the <b>suspendServer</b> script is run.
Resume_server	Listed when the <b>resumeServer</b> script is run.

### Undocumented VoiceXML server debugging feature

In some cases, being able to view the raw VoiceXML code generated by the server during application (try it out )visits can be very useful. In order to do that, you can take advantage of an undocumented VoiceXML debugging feature for the server.

To enable the VoiceXML debug mode, add the XML shown in Example 4-12 to the WEB-INF/web.xml file in the expanded directory from Audium.war on your application server, on the same level as the element named <load-on-startup>.

Example 4-12 XML to add to WEB-INF/web.xml file

```
<init-param>
<param-name>Debug</param-name>
<param-value>on</param-value>
</init-param>
```

Restart your application server for the change to take effect. Now, it will output VoiceXML pages sent as a response, as well as all of the incoming headers in the AUDIUM\_HOME/logs directory (not in application directories).

**Important:** This feature is for debugging use only in limited circumstances. Enabling this setting results in the creation of excessively large log files. Enabling this feature is *not recommended* on a production server, nor without having a good reason. The default is for it to be disabled under normal conditions.

## Debugging applications in the Web browser

This section describes how to debug applications using a web browser to simulate calls.

### *The first VoiceXML page*

To obtain the first VoiceXML page of an application, simply enter the URL a voice browser would access in a Web browser. If the Web browser appears on the same machine as Cisco CVP Server, the IP address Can Be Replaced With localhost. For Example, To Call The Helloworld Application In A Web browser, assuming that the application server listens on port 8080, you would enter:

```
http://localhost:8080/CVP/Server?application=HelloWorld
```

The Web browser then shows the first VoiceXML page.

### *Subsequent pages*

Because the Cisco CVP Server treats a phone call as a *session*, a cookie is used to keep track of it. The Web browser handled this cookie after the first VoiceXML page was returned. If you look at the cookies on the Web browser, you can see one for localhost. To view the subsequent VoiceXML pages, you must enter the next URL in the same browser window or a window that was derived from this window. For example, in Internet Explorer, choosing **File** → **New** → **Window** creates another window derived from the first. Clicking the IE icon in the taskbar does not.

To view subsequent VoiceXML pages, you must read the VoiceXML and find the <SUBMIT> that applies to the result you want to follow. Every <SUBMIT> will include various arguments that must be included in the URL in order to properly simulate a phone call. Simply place these arguments in the URL in between '&' characters. For example, if the VoiceXML contains the following submit:

```
<SUBMIT enctype="application/x-www-form-urlencoded" namelist="foundation_fld audium_vxmlLog confidence" method="post" next="/CVP/Server" />
```

The arguments are foundation\_fld, audium\_vxmlLog, and confidence. Let us say that in this example, the data captured and stored in foundation\_fld was apple and the confidence was 83%. The URL you would call would look something like:

```
http://localhost:8080/CVP/Server?foundation_fld=apple&confidence=0.83&audium_vxmlLog=my_logging_content
```

All submits will have the audium\_vxmlLog argument, which is used for interaction logging. You can leave this out if you want. Simply, there will be no interaction content stored in the activity log. The other data is required.

```
http://localhost:8080/CVP/Server?foundation_fld=apple&confidence=0.83
```

### *Hanging up*

As long as the Cisco CVP Server believes that a phone call is connected to it, a CVP Server port is used. Only when the phone call hangs up will the port become available again. When simulating the call in a Web browser, failing to simulate a hangup continues to occupy the

port until the internal timeout mechanism takes over (a default of 30 minutes). This can fill up quickly all the ports on Cisco CVP Server, necessitating a restart to quickly reset the ports. Therefore, when you simulate a phone call, you must simulate a hangup by calling the URL:

```
http://localhost:8080/CVP/Server?audium_action=hangup
```

After this URL is called, the Cisco CVP Server port is closed and the Web browser window can be used to simulate another call.

**Tips:** For debugging applications. keep these tips in mind:

- ▶ Do not use the Back and Forward buttons, they yield inconsistent results and most likely cause an error.
- ▶ To simulate a new call, it is best to open up a new browser window. Clicking the browser icon in the taskbar or quitting and starting the browser again will work. You can use the same window, but there can be some inconsistent behavior. Another tactic is to delete the localhost cookie.
- ▶ Only the first URL need to be include the application name, the Server knows the application name with the cookie.
- ▶ If you want to see the application root document, you must bring up the first VoiceXML page before the root document becomes available.
- ▶ To go quickly to a VoiceXML page deep in an application, open up a new browser window, then make many browser windows derive from it. Then start from the first window and enter the first URL, then enter the second URL in the second window, and so on. This process will allow you to go back through the same place for another phone call by simply selecting each window in turn and pressing the Reload button.

## Putting it all together: School Closure application

We finish this rather brief introduction to voice application development in the Cisco CVP environment by taking a look at a sample application that includes fairly advanced functionality and design principles. The application we examine is one that you might typically see deployed in a real-world self-service scenario. However, we only give you the bare essentials. We have not taken into account some typical hurdles that would typically crop up in the development of an application such as this. We leave the solving of these problems as an intellectual exercise for you later. We do, however, point out where some of these interesting conundrums might appear in the development process and give some salient hints on how to resolve these problems.

This is the School Closure information application. The idea is that parents wanting information about whether or not the school their children attend will be open or not during inclement weather, a public health emergency, or some other extraordinary event will call a number run by the state or local school district. At this number, they find a fairly sophisticated announcement service providing information about the status of the school for which they request information. This application is not so much interesting in that it incorporates a lot of elaborate business logic, but rather because it touches on some key design issues that allow for the introduction of rather more advanced application development techniques that should be useful to the large majority of developers now and in later work. Before we examine some of the more salient development techniques, we take a quick look at how we will expect our School Closure application to work as a whole. The business logic that drives this application is quite simple. Basically what happens in very broad strokes, is the following (pseudocode outline):

1. Determine the area code from which the user is calling.

2. If the user's area code can be determined from the call information, then ask the user if they are looking for information about a school within that area code. If so, go to 6 (this decision is not enabled in the demo application, we simply assume that we do not know the user's NPA and go through the whole process of retrieving necessary information from the user).
3. If the user does not want information about a school in the area code they are calling from (or area code information is not immediately available) ask the user which state they are looking for a school in (we use a custom element based on a Reusable Dialog Component to perform this task).
4. Ask the user in which city she or he is looking for a school. We will use a standard action element in conjunction with a Form\_With\_Confirm element to perform this action.
5. Ask which type of school for which she or he is looking (elementary, middle or high).
6. Finally, ask for the name of the school for which he or she would like to obtain information. We will use a custom Form\_With\_Confirm element built with the Java API to perform this step.
7. Query the database for information about the current status of the school and read relevant information back to the user.

Now clearly there are different ways that this application could be implemented in terms of business logic and user interface. We feel that this structure is effective and gets the job done. However, we encourage you to tweak this application as much as you desire to optimize performance, perform more stringent input validation, display more refined results, better search capability, and so forth. A step-by-step build of this rather complex application looks similar to this:

1. Before building the voice application we must do some setup behind the scenes. This application will be using a MySQL database as a back-end data repository. This means that we must complete the following before building and testing our application:
  - a. Install a MySQL database server. In the development environment, the database server can be coresident on the same machine as the application server. However, in a production environment, it is preferable to have the database running on a standalone server.
  - b. Setup relevant databases and tables using provided SQL scripts. We have provided a database that is populated with all public schools in the state of North Carolina, or you can simply use the scripts that define database and table structure and populate the relevant tables yourself).
  - c. Note that for the most part it is recommended to use a JNDI connection to a MySQL database for CVP projects. However, the configuration of a JNDI connection is somewhat cumbersome and complex, so for the purposes of this demonstration application we use an ODBC connection to MySQL instead. Create a system DSN on the application server where your application will run. You must install the MySQL ODBC driver in order to do so. Call the DSN `SchoolsDSN`. See the following URL to download the needed MySQL ODBC connector:
 

<http://dev.mysql.com/downloads/connector/odbc/3.51.html>
  - d. See the following URL for instructions on how to install MyODBC and configure a system DSN for use with our application:
 

<http://dev.mysql.com/doc/refman/5.0/en/installing-myodbc.html>
  - e. Create a Java project called `SchoolClosureInfoJavaClasses`. Add the necessary external jar files to your build path. This project will contain the classes that will be used in the `SchoolClosureInfo` voice project. Now create the `DBFunctionality` class with the source code shown in Example 4-13 . Remember to modify the user name and

password to whatever you specified when setting up your MySQL database server. This is a very rudimentary class that will allow you to run queries against a MySQL database.

*Example 4-13 SchoolClosureInfoJavaClasses*

---

```
/*
This class provides the workhorse DB functionality. Not terribly sophisticated or secure,
but it does the job we need it to do.
*/

import java.sql.*;

public class DBFunctionalityClass
{
    private Connection conn;

    public DBFunctionalityClass()
    {
        try
        {
            //load the odbc driver
            Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
            String dsn = "jdbc:odbc:SchoolsDSN";
            String user = "root";
            String password = "password";
            //Connect to the database
            conn = DriverManager.getConnection(dsn, user, password);
            conn.setAutoCommit(false);
        }
        catch(ClassNotFoundException e)
        {
            e.printStackTrace();
        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public ResultSet performQuery(String strSQL)
    {
        ResultSet rs = null;
        try
        {
            Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
            rs = stmt.executeQuery(strSQL);
        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        return rs;
    }
}
```



```

    public String performUpdate(String strSQL)
    {
        return "";
    }

    public String performDelete(String strSQL)
    {
        return "";
    }
}

```

2. We are going to build up this application in chunks. We will first build our initial section that collects the users area code from the ANI, and then the decision that determines whether or not we should proceed based on a known or unknown area code. Create a new project call SchoolClosureInfo. Add the following elements to the call flow: an action element and a decision element as shown in Figure 4-188.

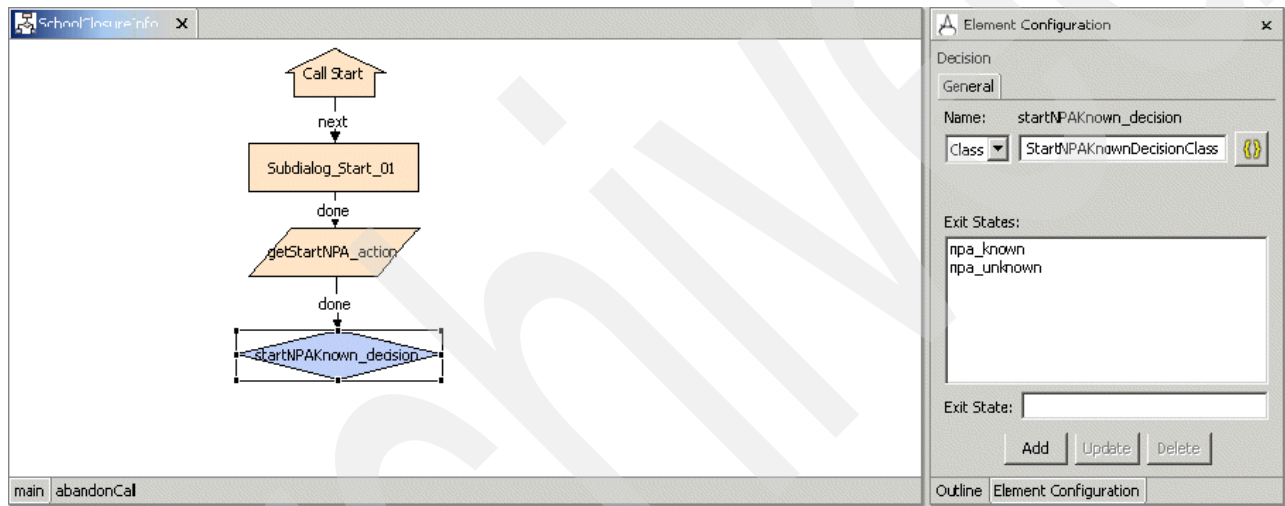


Figure 4-188 SchoolClosureInfo Initial call flow section

The getStartNPA\_action standard element will have a class called GetStartNPAActionClass that defines its functionality. The functionality is extremely simple. All this action element does is take the user's ANI and attempts to extract the user's area code by taking the first three digits of the user's ANI. All that you must do to implement this functionality is to create the GetStartNPAActionClass.java file in the java project and add the code shown in Example 4-14 to it.

*Example 4-14 GetStartNPAActionClass.java*

```

/*
This class is first one executed, extracts NPA from ANI if possible
*/

import com.audium.server.AudiumException;
import com.audium.server.session.ActionElementData;
import com.audium.server.voiceElement.ActionElementBase;

public class GetStartNPAActionClass extends ActionElementBase
{
    public void doAction(String name, ActionElementData actionData) throws AudiumException

```

```

    {
        String userNPA = getUserNPA(actionData.getAni());
        if(!userNPA.equalsIgnoreCase("NA"))
        {
            actionData.setSessionData("userNPA", userNPA);
        }
        else
        {
            actionData.setSessionData("userNPA", "");
        }
    }

    private String getUserNPA(String userANI)
    {
        //this method takes an ANI as a parameter, parses out the NPA
        //(area code) and returns it to the calling routine
        String NPA = userANI.substring(0, 2);
        return NPA;
    }
}

```

---

The idea here is that the area code will be extracted from the user's ANI and stored in session data for later use. Once the code for the class has been entered into the java source file, rebuild the Java project. Now, copy the generated `GetStartNPAAActionClass.class` file to the `/deploy/java/application/classes` folder in the voice project.

The `startNPAKnown_decision` will simply determine whether or not the `getStartNPA_action` element was able to determine a usable area code. We will define two possible exit states for this element: `npa_known` and `npa_unknown`. The decision-making logic will be implemented in a Java class called `StartNPAKnownDecisionClass` with the source code shown in Example 4-15.

---

*Example 4-15 StartNPAKnownDecisionClass*

---

```

/*

Just a check to see if we are starting with a known NPA or not

*/

import com.audium.server.AudiumException;
import com.audium.server.voiceElement.*;
import com.audium.server.session.DecisionElementData;

public class StartNPAKnownDecisionClass extends DecisionElementBase
{
    public String doDecision(String name, DecisionElementData decisionData) throws
AudiumException
    {
        String exitState = "";
        if(decisionData.getSessionData("userNPA").toString().equals(""))
        {
            exitState = "npa_known";
        }
        else
        {

```

```

        exitState = "npa_unknown";
    }
    return exitState;
}
}

```

The logic for the decision should be fairly obvious. Make sure to copy the generated class file to the java deployment folder in the voice project as well. For the purposes of our demonstration application, we simply assume that the user's NPA is unknown and collect all necessary information from the user to figure out for which school the user would like information.

3. Now let us start developing the branch of the application that becomes active when the users NPA is unknown. Configure startNPAknown\_decision so that it is skipped and exits with the "npa\_unknown" state. See Figure 4-189.

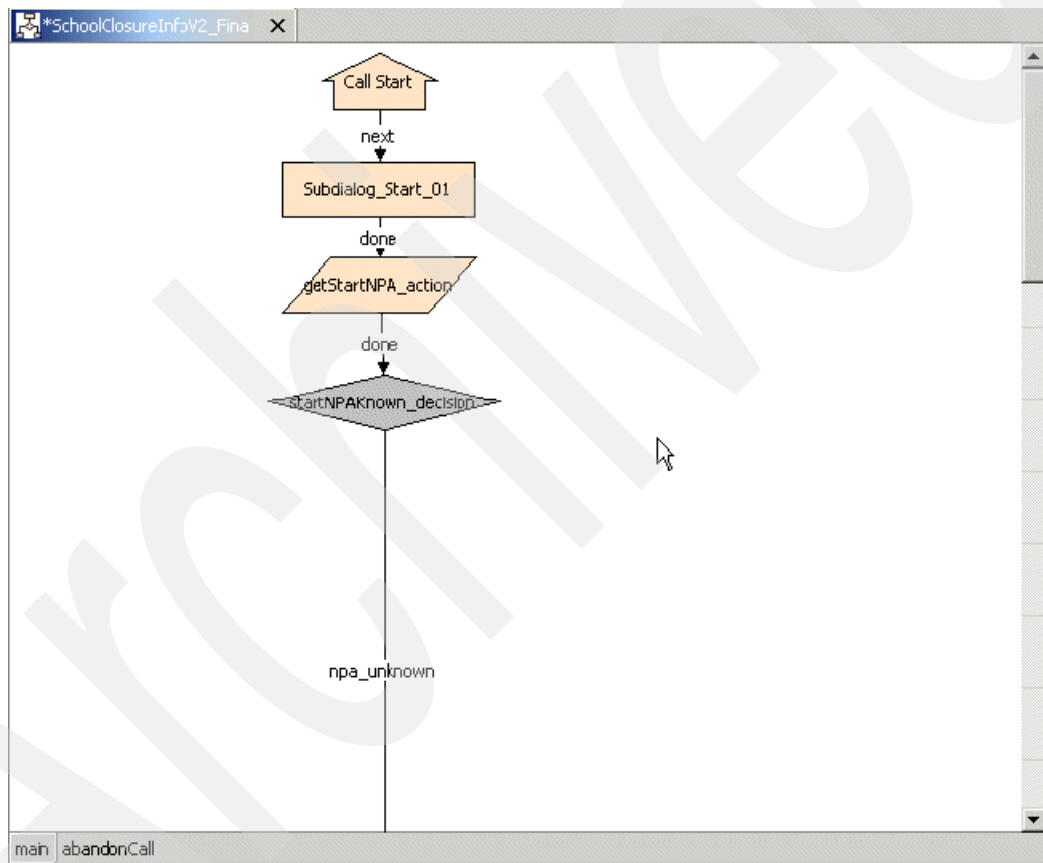


Figure 4-189 SchoolClosureInfo: Branch to be active when users NPA is unknown

4. While we are at it, create a separate page called abandonCall that handles application hang-ups (see Figure 4-190). This functionality will be used in several places in the application, so make a reusable page to encapsulate it.

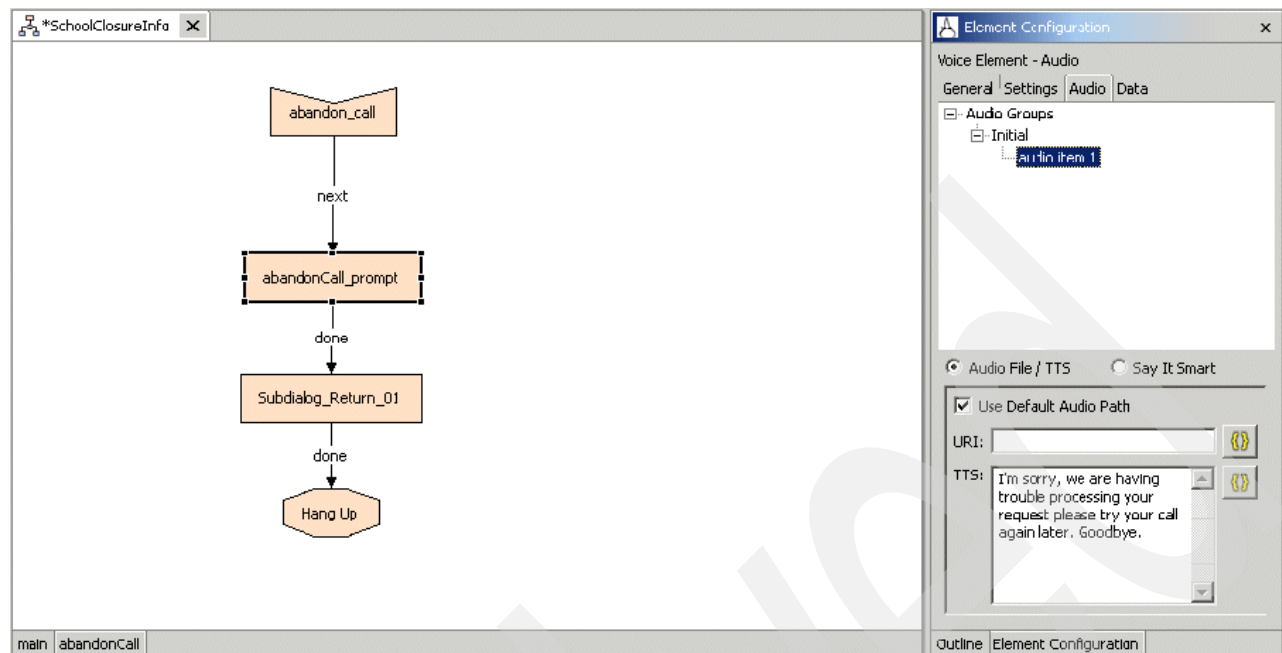


Figure 4-190 SchoolClosureInfo: abandonCall page

5. Next, We want to ask the user in which state the school for which the user would like information is located. This could be accomplished quite easily using a standard Form\_With\_Confirm element coupled with either an inline or an external grammar with the names of all 50 states. We do something quite a bit more interesting, however. We are going to use a custom element built using the states Reusable Dialog Component. For more information about RDCs, see the following URL. Here are instructions on how we would integrate the US state RDC into our call flow. Note that similar means can be used to integrate any other RDC component:

<http://jakarta.apache.org/taglibs/sandbox/doc/rdc-doc/intro.html>

- a. You must first create an RDC-enabled Web application. Instructions on how to do that are outside the scope of this tutorial, but you can obtain very clear instructions on doing that at the following URL and call-usState.jsp files in your application. Include the usStateRDC.vxml.

<http://jakarta.apache.org/taglibs/sandbox/doc/rdc-doc/index.html>

- b. Edit usStateRDC.vxml by making these changes:

- Replace all occurrences of the string `http://192.168.162.43:8080` with the URL of Cisco CVP server.
- Replace all occurrences of the string `http://192.168.160.43:7043/rdc-examples/` with the server + path of the Web application with the RDC taglib installed.
- Replace all occurrences of the string `RDCTest` with the name of the application this RDC is used in, for example `SchoolClosureInfo`.

- c. Copy your edited versions of `usStateRDC.vxml` (see Example 4-16) and `call-usState.jsp` (see Example 4-17 on page 274) into the root directory of the RDC-enabled web application. For example, if it's installed in `tomcat/webapps/myappwithRDCs/`, then put these two files directly into that directory. In our tests, this directory was `tomcat/webapps/rdc-examples/` because we used the `rdc-examples` application that comes with the RDCs as my RDC taglib enabled application.

- d. Add a VoiceXML Insert element to the CVP Studio callflow and configure it (see Figure 4-191 on page 275):
  - Add one exit state: done. For simplicity's sake, no other exit states are implemented. The RDCs can return max no match, max no input, and so forth. However, these are not being handled in this simple wrapper.
  - Specify the URI of the usStateRDC.vxml file you have now hosted on the app server as the source for the VoiceXML insert element.
- e. Add an Audio element (call it sayStateName\_audio) before the VoiceXML Insert element that includes the prompt, for example, What state is the desired school in? The RDC will follow that, when it is visited, with Say the US state.
- f. To access the state that the user said, as a two-letter abbreviation, access the element data of the VoiceXML Insert; it is in the variable named state.

*Example 4-16 usStateRDC.vxml*

---

```

<?xml version="1.0"?>
<vxml version="2.0"
application="http://192.168.162.43:8080/CVP/Server?audium_vxml_root=true&calling_into=R
DCTest&namelist=element_log_state">
  <form>
    <!-- Assign values to CVP Server required variables -->
    <block>
      <assign name="audium_element_start_time_millisecs" expr="new Date().getTime()" />
      <assign name="audium_vxmlLog" expr="'|||prompt$$$initial_prompt^^^' +
application.getElapsedTime(audium_element_start_time_millisecs)"/>
    </block>
    <!-- As a subdialog, call the JSP page that calls the usState RDC -->
    <subdialog name="stateSubdialog"
src="http://192.168.160.43:7043/rdc-examples/call-usState.jsp">
      <filled>
        <!-- When the subdialog is done, extract the US state from the resulting object -->
        <script src="http://192.168.160.43:7043/rdc-examples/.grammar/return.js"/>
        <var name="element_log_state" expr="deserializeReturnValue(stateSubdialog.state)"/>
        <assign name="element_log_state" expr="element_log_state.getValue()"/>
        <assign name="audium_vxmlLog" expr="audium_vxmlLog +
'||||finished$$$finished_page^^^' +
application.getElapsedTime(audium_element_start_time_millisecs)"/>
        <assign name="audium_exit_state" expr="'done'"/>
        <!-- Return control to CVP Server, passing along the chosen state and required CVP
Server variables -->
        <return namelist="audium_exit_state audium_vxmlLog audium_error audium_hotlink
audium_hotevent audium_action element_log_state" />
      </filled>
    </subdialog>
  </form>
</vxml>

```

---

*Example 4-17 callUSState.jsp*

---

```

<!-- Calls the usState RDC as a subdialog -->
<!--
<%@ page language="java" contentType="application/voicexml+xml" %>
<%@ taglib prefix="rdc" uri="http://jakarta.apache.org/taglibs/rdc-1.0"%>
-->
<vxml version="2.0">
  <jsp:useBean id="dialogMap" class="java.util.LinkedHashMap" scope="session"/>
  <rdc:task map="{dialogMap}">
    <rdc:usState id="state" confirm="true"/>
  </rdc:task>
</vxml>

```

---

```

<block>
  <var name="state" expr="'${state}'"/>
  <return namelist="state"/>
</block>
</rdc:task>
</vxml>

```

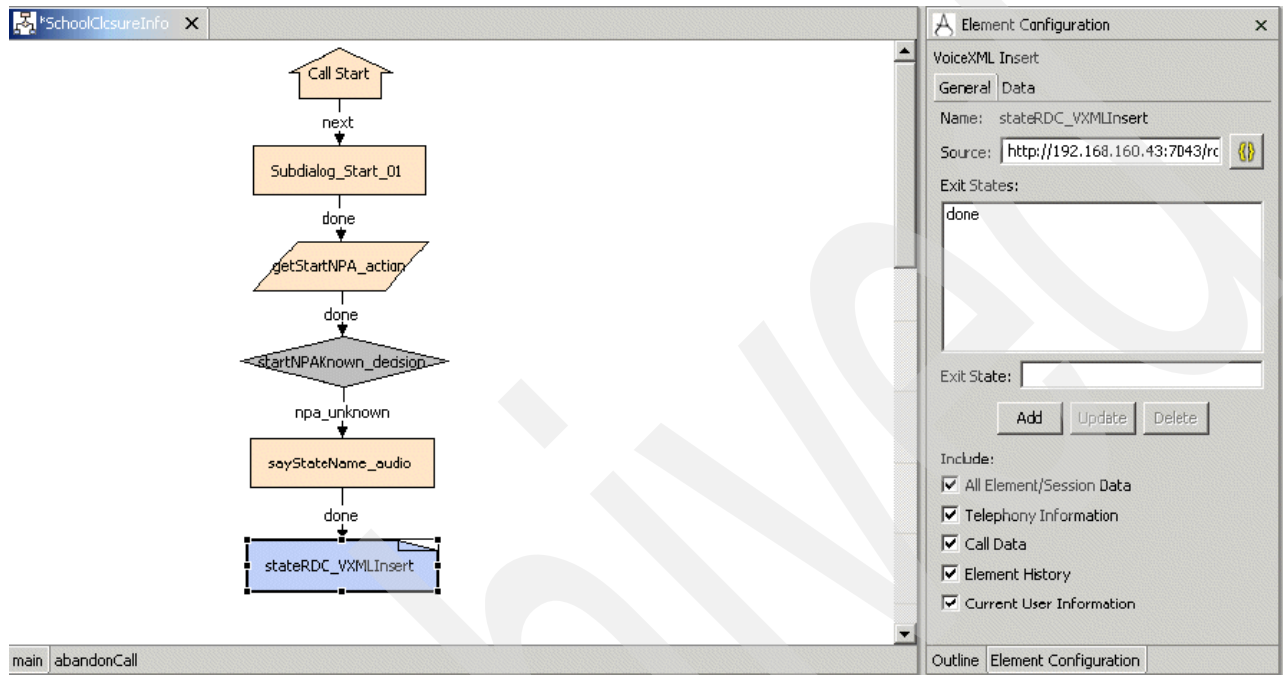


Figure 4-191 SchoolClosureInfo: stateRDC\_VXMLInsert element

The name of the state selected by the user is now be stored in the state element variable for the stateRDC\_VXMLInsert element.

- For the sake of consistency, we put this into session data (selectedState will be the variable). Strictly speaking, this is not necessary because it would be just as simple to access the element data. However, you will find as time goes on and the complexity of your projects increases, that it is almost always better if you can consolidate data storage into a consistent location. Session data is particularly handy for this purpose. Create the putStateNameIntoSession\_action element (see Figure 4-192 on page 276) and a corresponding Java class (putStateNameIntoSessionActionClass) with the source code shown in Example 4-18.

#### Example 4-18 PutStateNameIntoSessionActionClass

```

/*
 * This class simply takes a piece of element data and stores it in session data
 */
import com.audium.server.AudiumException;
import com.audium.server.session.ActionElementData;
import com.audium.server.voiceElement.ActionElementBase;
import java.sql.*;

```

```

public class PutStateNameIntoSessionActionClass extends ActionElementBase
{
    public void doAction(String name, ActionElementData actionData) throws AudiumException
    {
        String state = actionData.getElementData("stateRDC_VXMLInsert", "value").toString();
        actionData.setSessionData("selectedState", schoolId);
    }
}

```

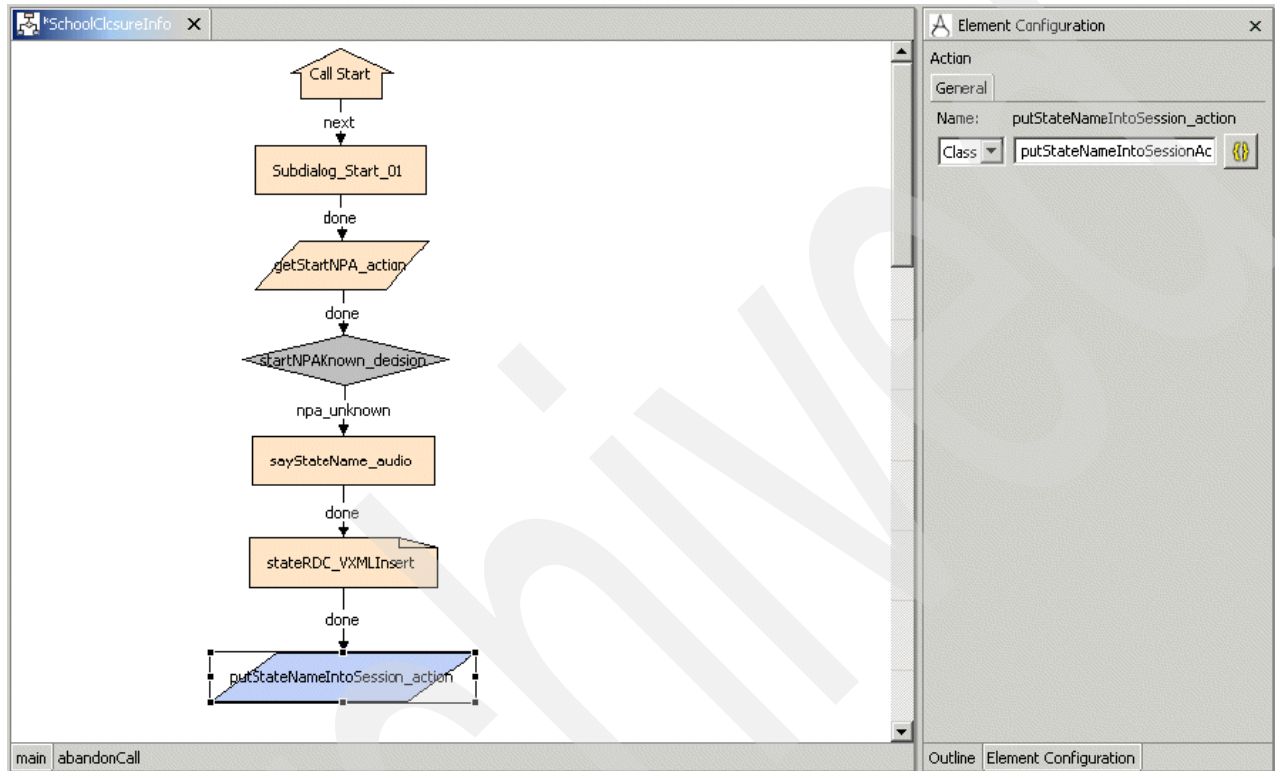


Figure 4-192 SchoolClosureInfo: putStateNameIntoSession\_action element

- Now, we know the name of the state in which we will be searching for the school. We use a custom action element to create a grammar dynamically, containing the names of all cities in the selected state.

**Note:** This is not the most efficient way to create an external grammar dynamically. This method is quite taxing on the application server in terms of disk I/O usage when creating a temporary grammar file for every active session. But we show it simply to illustrate how you might approach such a development task. Later, we show you a far more efficient means of doing the same thing using a component constructed using the Java API. First we create an action element called createCityGrammar\_action and an associated java class CreateCityGrammarActionClass as shown in Figure 4-193 on page 277.



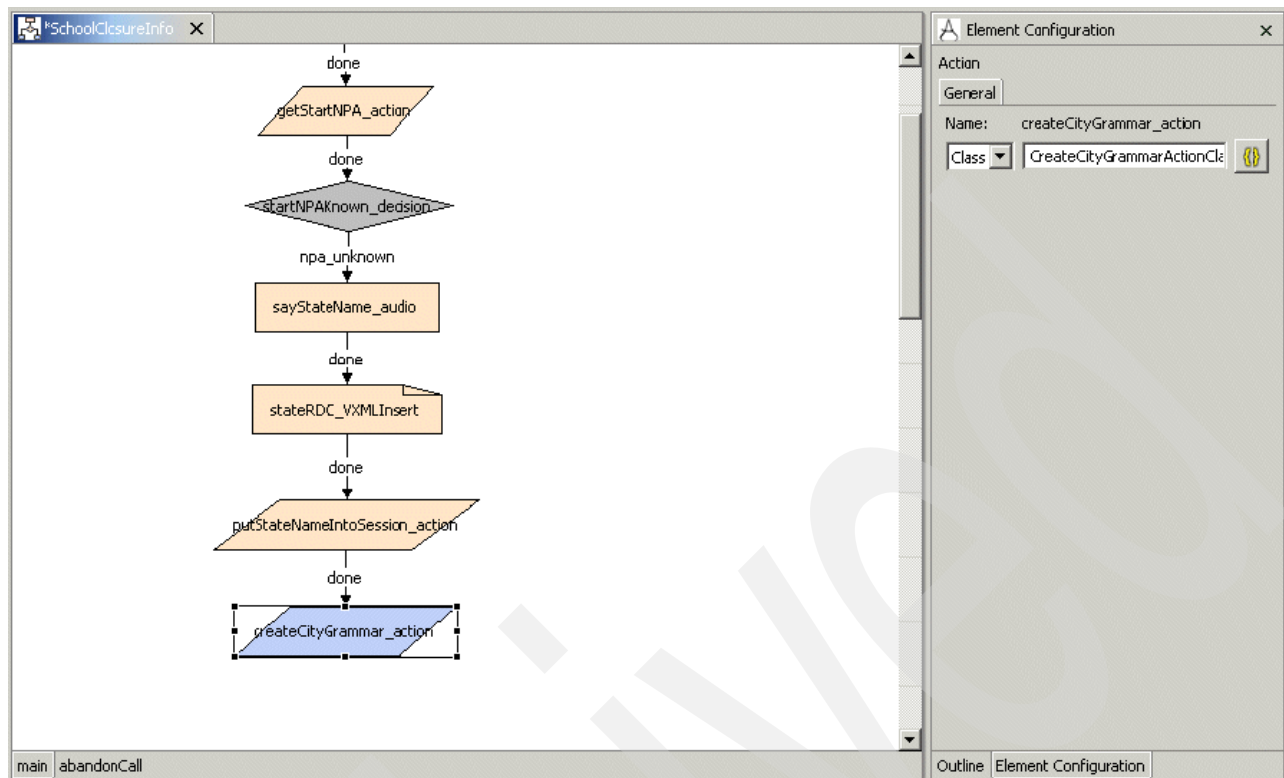


Figure 4-193 SchoolClosureInfo: createCityGrammar\_action element

Example 4-19 shows the source code for the java class.

#### Example 4-19 CreateCityGrammarActionClass

```
/*

This class creates a grammar listing all cities for a given state
that have school in the database

*/

import com.audium.server.AudiumException;
import com.audium.server.session.ActionElementData;
import com.audium.server.voiceElement.ActionElementBase;
import java.sql.*;
import java.io.*;
import java.util.*;

public class CreateCityGrammarActionClass extends ActionElementBase
{
    public void doAction(String name, ActionElementData actionData) throws AudiumException
    {
        String selectedState = actionData.getSessionData("selectedState").toString();
        if(selectedState.length() != 2)
        {
            //if the two char abbreviation for the state is not given
            selectedState = getStateAbbreviation(selectedState);
        }
        DBFunctionalityClass dbObj = new DBFunctionalityClass();
        ArrayList cityNames = new ArrayList();
    }
}
```

```

try
{
    String strSQL = "SELECT DISTINCT City FROM schools_tb WHERE State = '" +
selectedState + "'";
    ResultSet rs = dbObj.performQuery(strSQL);
    if(rs != null)
    {
        boolean more = rs.next();
        while(more)
        {
            String cName = rs.getString("City");
            cityNames.add(cName);
            more = rs.next();
        }
        rs.close();
    }
}
catch(SQLException e)
{
    e.printStackTrace();
}

//now create a unique file name for the dynamically generated grammar
String grammarFilePath = "C:\\Cisco\\CVP\\Tomcat 4.1\\webapps\\CVP\\grammars\\" +
createUniqueGrammarName(actionData);
//now generate the grammar file itself
String grammarFileContents = generateGrammarFileContents(cityNames);
//store the grammar file to the grammar directory on the app server
if(storeDynamicGrammar(grammarFileContents, grammarFilePath))
{
    //save the path to the grammar on the server in the "" session variable
    //so we can get rid of this temp file later
    String grammarURL = "http://9.42.171.24:8080/CVP/grammars/" +
createUniqueGrammarName(actionData);
    actionData.setSessionData("citiesTempGrammar", grammarURL);
}
else
{
    //storing of our dynamic grammar has failed
    //use the default grammar
}
}

private String getStateAbbreviation(String fullStateName)
{
    //this method takes a full state name as a param and return the abbreviation
    String abbreviation = "";
    if(fullStateName.equalsIgnoreCase("ALABAMA"))
    {
        abbreviation = "AL";
    }
    else if(fullStateName.equalsIgnoreCase("ALASKA"))
    {
        abbreviation = "AK";
    }
    else if(fullStateName.equalsIgnoreCase("ARIZONA"))
    {
        abbreviation = "AZ";
    }
    else if(fullStateName.equalsIgnoreCase("ARKANSAS"))
    {

```

```

        abbreviation = "AR";
    }
    else if(fullStateName.equalsIgnoreCase("CALIFORNIA"))
    {
        abbreviation = "CA";
    }
    else if(fullStateName.equalsIgnoreCase("COLORADO"))
    {
        abbreviation = "CO";
    }
    else if(fullStateName.equalsIgnoreCase("CONNECTICUT"))
    {
        abbreviation = "CT";
    }
    else if(fullStateName.equalsIgnoreCase("DELAWARE"))
    {
        abbreviation = "DE";
    }
    else if(fullStateName.equalsIgnoreCase("DISTRICT OF COLUMBIA") ||
fullStateName.equalsIgnoreCase("D C"))
    {
        abbreviation = "DC";
    }
    else if(fullStateName.equalsIgnoreCase("FLORIDA"))
    {
        abbreviation = "FL";
    }
    else if(fullStateName.equalsIgnoreCase("GEORGIA"))
    {
        abbreviation = "GA";
    }
    else if(fullStateName.equalsIgnoreCase("HAWAII"))
    {
        abbreviation = "HI";
    }
    else if(fullStateName.equalsIgnoreCase("IDAHO"))
    {
        abbreviation = "ID";
    }
    else if(fullStateName.equalsIgnoreCase("ILLINOIS"))
    {
        abbreviation = "IL";
    }
    else if(fullStateName.equalsIgnoreCase("INDIANA"))
    {
        abbreviation = "IN";
    }
    else if(fullStateName.equalsIgnoreCase("IOWA"))
    {
        abbreviation = "IA";
    }
    else if(fullStateName.equalsIgnoreCase("KANSAS"))
    {
        abbreviation = "KS";
    }
    else if(fullStateName.equalsIgnoreCase("KENTUCKY"))
    {
        abbreviation = "KY";
    }
    else if(fullStateName.equalsIgnoreCase("LOUISIANA"))

```

```

{
    abbreviation = "LA";
}
else if(fullStateName.equalsIgnoreCase("MAINE"))
{
    abbreviation = "ME";
}
else if(fullStateName.equalsIgnoreCase("MARYLAND"))
{
    abbreviation = "MD";
}
else if(fullStateName.equalsIgnoreCase("MASSACHUSETTS"))
{
    abbreviation = "MA";
}
else if(fullStateName.equalsIgnoreCase("MICHIGAN"))
{
    abbreviation = "MI";
}
else if(fullStateName.equalsIgnoreCase("MINNESOTA"))
{
    abbreviation = "MN";
}
else if(fullStateName.equalsIgnoreCase("MISSISSIPPI"))
{
    abbreviation = "MS";
}
else if(fullStateName.equalsIgnoreCase("MISSOURI"))
{
    abbreviation = "MO";
}
else if(fullStateName.equalsIgnoreCase("MONTANA"))
{
    abbreviation = "MT";
}
else if(fullStateName.equalsIgnoreCase("NEBRASKA"))
{
    abbreviation = "NE";
}
else if(fullStateName.equalsIgnoreCase("NEVADA"))
{
    abbreviation = "NV";
}
else if(fullStateName.equalsIgnoreCase("NEW HAMPSHIRE"))
{
    abbreviation = "NH";
}
else if(fullStateName.equalsIgnoreCase("NEW JERSEY"))
{
    abbreviation = "NJ";
}
else if(fullStateName.equalsIgnoreCase("NEW MEXICO"))
{
    abbreviation = "NM";
}
else if(fullStateName.equalsIgnoreCase("NEW YORK"))
{
    abbreviation = "NY";
}
else if(fullStateName.equalsIgnoreCase("NORTH CAROLINA"))

```

```

{
    abbreviation = "NC";
}
else if(fullStateName.equalsIgnoreCase("NORTH DAKOTA"))
{
    abbreviation = "ND";
}
else if(fullStateName.equalsIgnoreCase("OHIO"))
{
    abbreviation = "OH";
}
else if(fullStateName.equalsIgnoreCase("OKLAHOMA"))
{
    abbreviation = "OK";
}
else if(fullStateName.equalsIgnoreCase("OREGON"))
{
    abbreviation = "OR";
}
else if(fullStateName.equalsIgnoreCase("PENNSYLVANIA"))
{
    abbreviation = "PA";
}
else if(fullStateName.equalsIgnoreCase("RHODE ISLAND"))
{
    abbreviation = "RI";
}
else if(fullStateName.equalsIgnoreCase("SOUTH CAROLINA"))
{
    abbreviation = "SC";
}
else if(fullStateName.equalsIgnoreCase("SOUTH DAKOTA"))
{
    abbreviation = "SD";
}
else if(fullStateName.equalsIgnoreCase("TENNESSEE"))
{
    abbreviation = "TN";
}
else if(fullStateName.equalsIgnoreCase("TEXAS"))
{
    abbreviation = "TX";
}
else if(fullStateName.equalsIgnoreCase("UTAH"))
{
    abbreviation = "UT";
}
else if(fullStateName.equalsIgnoreCase("VERMONT"))
{
    abbreviation = "VT";
}
else if(fullStateName.equalsIgnoreCase("VIRGINIA"))
{
    abbreviation = "VA";
}
else if(fullStateName.equalsIgnoreCase("WASHINGTON"))
{
    abbreviation = "WA";
}
else if(fullStateName.equalsIgnoreCase("WEST VIRGINIA"))

```

```

        {
            abbreviation = "WV";
        }
        else if(fullStateName.equalsIgnoreCase("WISCONSIN"))
        {
            abbreviation = "WI";
        }
        else if(fullStateName.equalsIgnoreCase("WYOMING"))
        {
            abbreviation = "WY";
        }
        return abbreviation;
    }

    private String createUniqueGrammarName(ActionElementData aD)
    {
        //unique name will have the following form: <session id>_init.grxml
        String uniqueName = aD.getSessionId().replaceAll("9.42.171.24.",
        "").replaceAll("SchoolClosureInfo", "").substring(0, 12) + "_cities.grxml";
        //debug
        //System.out.println(uniqueName);
        //
        return uniqueName;
    }

    private String generateGrammarFileContents(ArrayList schoolNames)
    {
        //add code for optional keywords here...
        String contents = "<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>";
        contents += "<!DOCTYPE grammar PUBLIC \"-//W3C//DTD GRAMMAR 1.0//EN\"";
        contents += "\"http://www.w3.org/TR/speech-grammar/grammar.dtd\">";
        contents += "<grammar version=\"1.0\" xmlns=\"http://www.w3.org/2001/06/grammar\"";
        contents += "tag-format=\"semantics/1.0\" xml:lang=\"en-US\" mode=\"voice\" root=\"main_rule\">";
        contents += "<rule id=\"main_rule\" scope=\"public\">";
        contents += "<one-of>";
        for(int i = 0; i <= schoolNames.size() - 1; i++)
        {
            contents += "<item>" + schoolNames.get(i) + "</item>";
        }
        contents += "</one-of>";
        contents += "</rule>";
        contents += "</grammar>";
        return contents;
    }

    private boolean storeDynamicGrammar(String grammarFileContents, String grammarFilePath)
    {
        try
        {
            //save the file to the grammars directory on the app server
            File oFile = new File(grammarFilePath);
            FileWriter fW = new FileWriter(oFile);
            fW.write(grammarFileContents);
            fW.close();
            return true;
        }
    }

```

```

        catch(Exception e)
        {
            //e.printStackTrace();
            return false;
        }
    }
}

```

Now, this class will create a temporary grammar file with a name generated based on the current session ID (to ensure uniqueness of the file name) containing all the cities in the database for a given state. The name and location of the temporary grammar file is stored in session variable `Data.Session.citiesTempGrammar`. Next, we create a form that will use this grammar by setting the Voice Grammar setting to the value of the session variable as shown in Figure 4-194 on page 283.

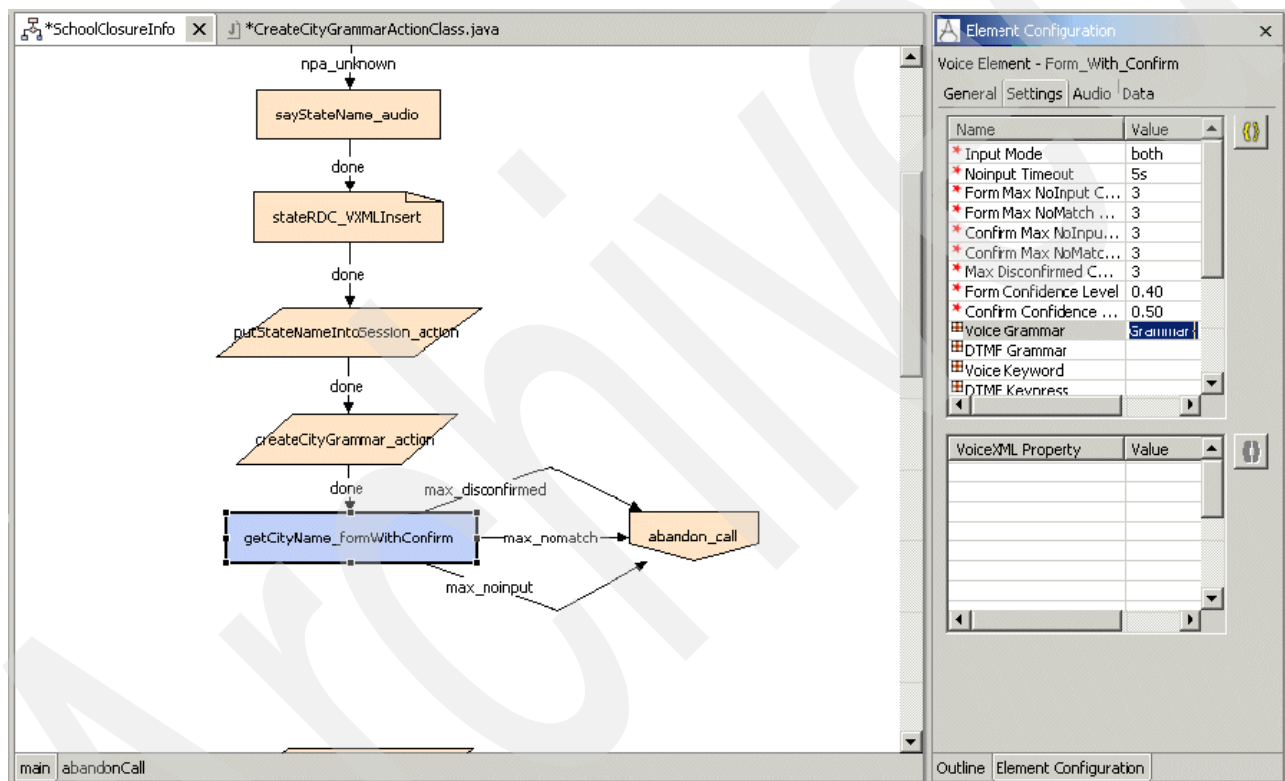


Figure 4-194 *SchoolClosureInfo: getCityName\_formWithConfirm element*

- After we have obtained the name of the city, we must clean up the temporary external grammar file we have just created. Create an action element called `cleanUpTempGrammars_action` and the associated Java class `CleanUpTempGrammarsActionClass` as shown in Figure 4-195 on page 284.



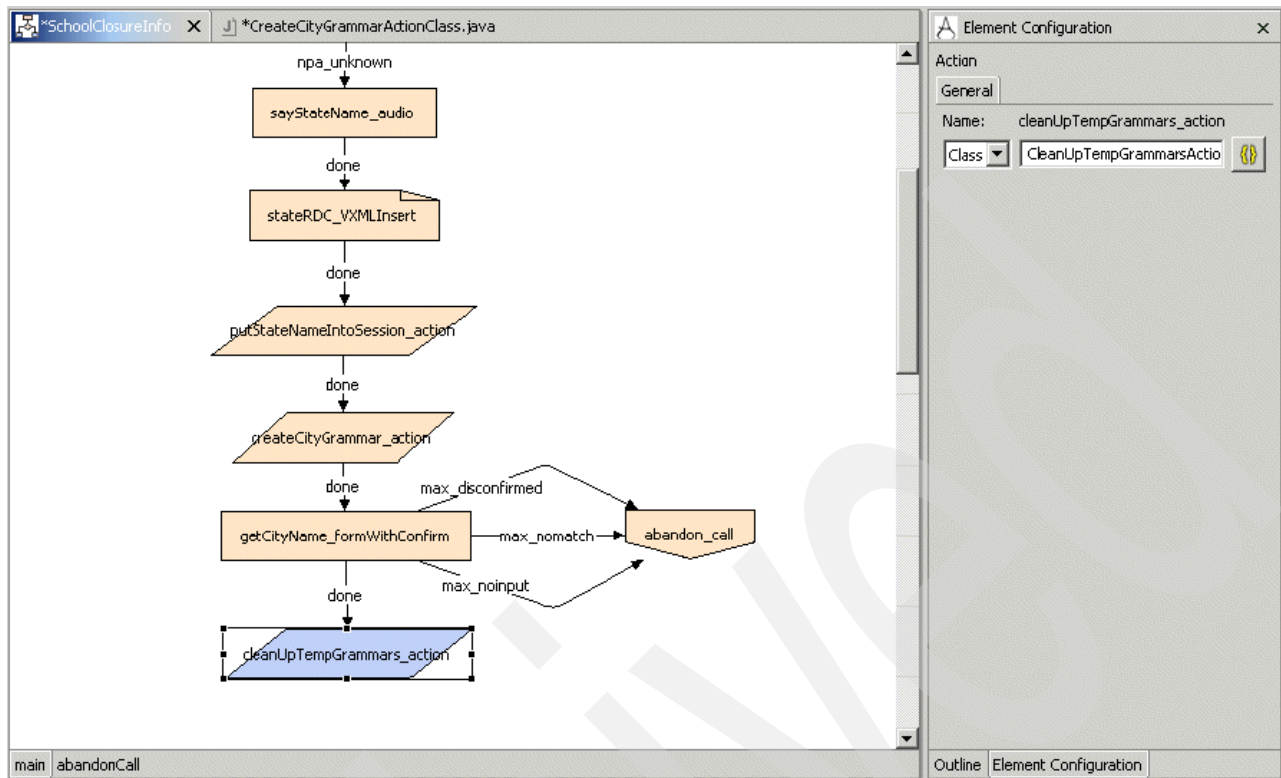


Figure 4-195 SchoolClosureInfo: cleanUpTempGrammars\_action element

CleanUpTempGrammarsActionClass will have the source code shown in Example 4-20.

**Note:** We used an array (filesToDelete) to hold the names of temp files that must be deleted. This is in case we decide to generalize this component at some point in the future so that it can take care of the deletion of multiple temp files if appropriate.

#### Example 4-20 CleanUpTempGrammarsActionClass

```

/*
This class simply removes the dynamically created grammars that
that are no longer needed.
*/

import com.audium.server.AudiumException;
import com.audium.server.session.ActionElementData;
import com.audium.server.voiceElement.ActionElementBase;
import java.io.*;

public class CleanUpTempGrammarsActionClass extends ActionElementBase
{
    public void doAction(String name, ActionElementData actionData) throws AudiumException
    {
        String[] filesToDelete = new String[1];
        filesToDelete[0] = (String)actionData.getSessionData("citiesTempGrammar");
        try
        {
            for(int i = 0; i <= filesToDelete.length - 1; i++)

```

```

    {
        try
        {
            String fullPath = "C:\\Cisco\\CVP\\Tomcat 4.1\\webapps\\CVP\\grammars\\" +
filesToDelete[i].replaceAll("http://9.42.171.24:8080/CVP/grammars/", "");
            File f = new File(fullPath);
            f.delete();
        }
        catch(Exception e)
        {
            //if file doesn't exist just catch and move on to next array element
        }
    }
}
catch(Exception e)
{
    e.printStackTrace();
}
}
}

```

9. At this point, we have cleaned up our temp file and we also know the state the school is located in and the city the school is located in. We must now determine whether the user is looking for information about an elementary, middle or high school. Why do we need to know this? The simple fact is that in many municipalities, there are multiple schools with the same name, for example. Gardner Elementary and High Schools. Asking the type of school helps ensure that we retrieve the right information for the right school). This part is easy. Simply define a form with an inline grammar listing the three school types: elementary, middle and high school. Please refer to Figure 4-196 on page 285.

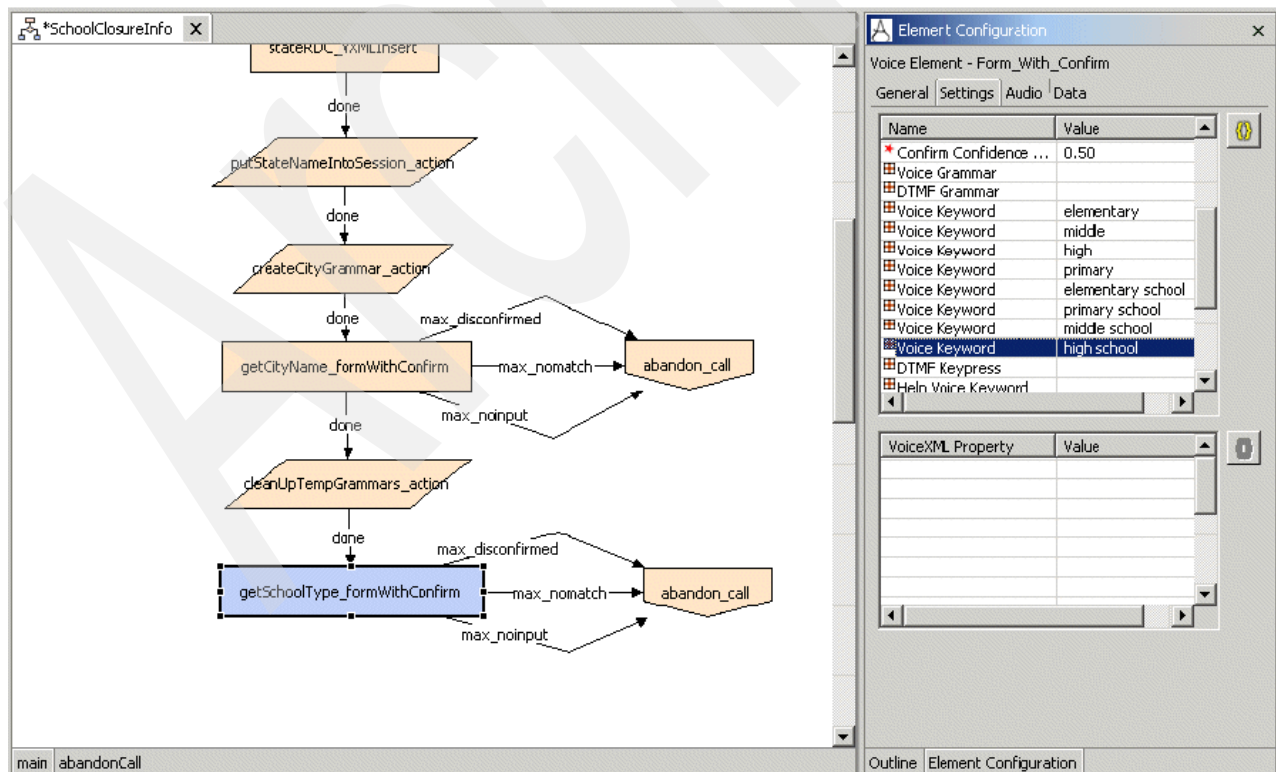


Figure 4-196 SchoolClosureInfo: getSchoolType\_formWithConfirm element

10. We now know the state, city and type of school for which we are looking. The last thing we must know from the user is the name of the school. It is important to note that in the vast majority of cases, these four pieces of information are enough to narrow the user selection down to a single school. However, this is most certainly not always the case. In a real-world deployment of this application, you would be required to account for the fact that sometimes there will be two schools of the same type, with the same name located in the same city. To identify the correct school, you must first determine whether or not the information provided by the user is sufficient to identify a single school. If not, you must request further information from the user, perhaps a street name or a zip code. We leave the development of a search validation and refinement mechanism for you as a further exercise.

For now, we work on the assumption that the four pieces of information provided by the user are sufficient to identify a unique school. We now must determine the name of the school for which we are looking. Earlier, we created a dynamic grammar of city names based on the state the user supplied. Recall that we implemented the dynamic grammar generation with the caveat that the chosen implementation would cause a severe impact on disk I/O on the application server.

Here we introduce the idea of a custom configurable component constructed entirely using the java API that does not require creating temporary external grammar files on the application server, but that instead relies on the creation of a dynamic *inline* grammar. The source code shown in Example 4-21 on page 287 is for a form element that dynamically creates an inline grammar of school names for itself by querying the schools database based on criteria retrieved from session data in the voice application. First build the class file in the Java project, then copy the class file over to the java deployment folder in the voice project. Save and close the call flow, then reopen it. Notice in the elements pane that something new has appeared. Under the local elements folder, we now see our School\_Name\_With\_Confirm form element available to be dragged and dropped into our call flow as shown in Figure 4-197.

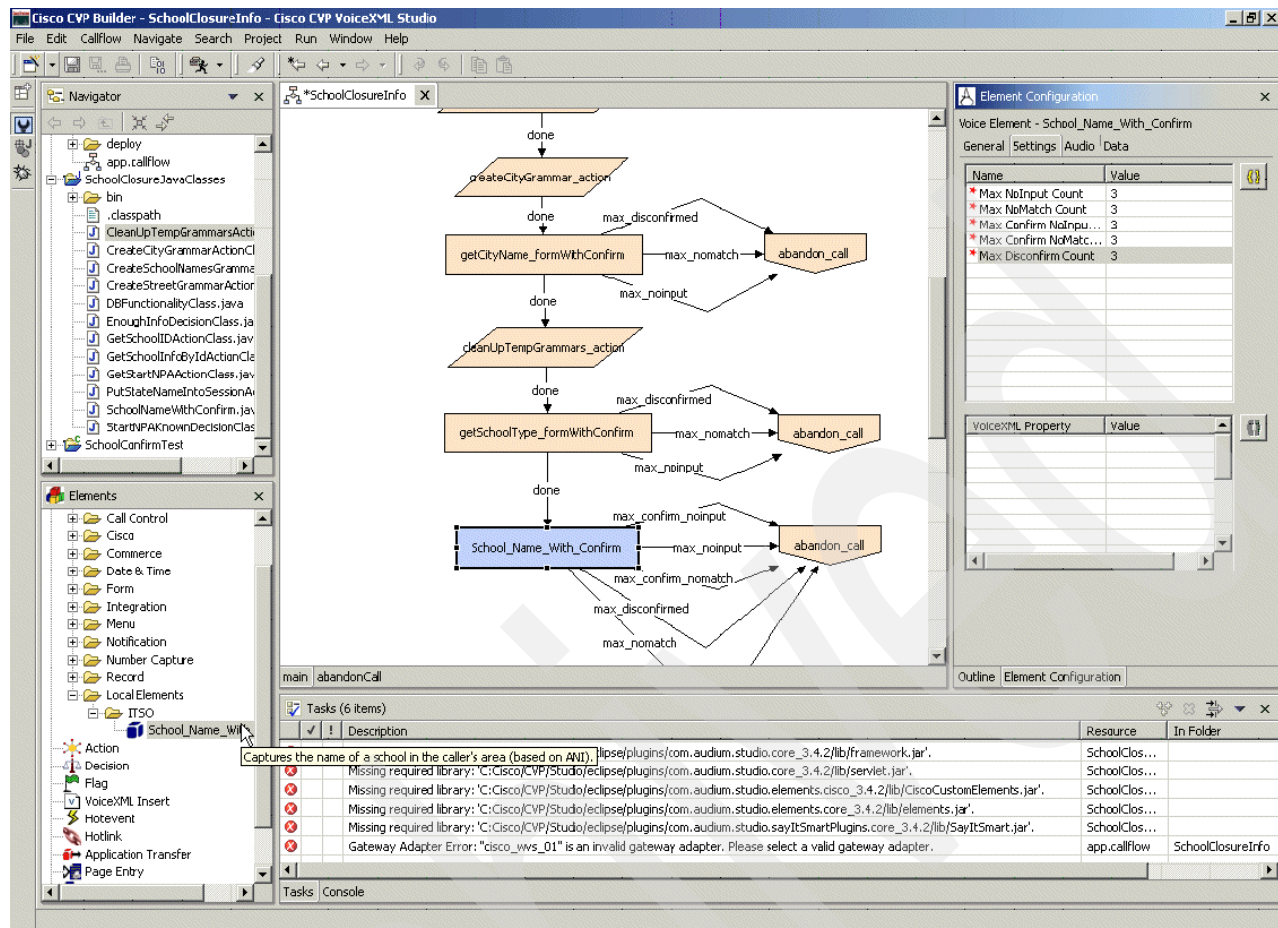


Figure 4-197 SchoolClosureInfo: School\_Name\_with\_Confirm element

Example 4-21 shows the source code for the custom form component.

#### Example 4-21 SchoolNameWithConfirm

```
import java.util.*;

// These classes are used by custom elements.
import com.audium.server.voiceElement.VoiceElementBase;
import com.audium.server.voiceElement.ElementInterface;
import com.audium.server.voiceElement.Setting;
import com.audium.server.voiceElement.ExitState;
import com.audium.server.voiceElement.ElementData;
import com.audium.server.voiceElement.ElementException;

// This class is used by voice elements.
import com.audium.server.session.VoiceElementData;
import com.audium.server.voiceElement.AudioGroup;
import com.audium.server.xml.VoiceElementConfig;
import com.audium.server.AudiumException;

// Include these to use VFCs.
import com.audium.core.vfc.*;
import com.audium.core.vfc.util.*;
import com.audium.core.vfc.form.VForm;
import com.audium.core.vfc.audio.VAudio;
import com.audium.core.vfc.form.VBuiltInField;
```

```

import com.audium.core.vfc.form.VMenuField;
import com.audium.core.vfc.util.VoiceInput;

// Used for DB access
import java.sql.*;

/**
 * This defines a custom voice element. The marker class ElementInterface is
 * needed so it will show up in Audium Builder.
 */
public class SchoolNameWithConfirm extends VoiceElementBase implements ElementInterface
{
    /**
     * This method is where the voice element is expected to produce the
     * appropriate VoiceXML in the form of VFCs. The VMain object is where the
     * VFCs are to be added to, the Hashtable contains the submit arguments
     * sent by the voice browser (or null if there are none), and
     * VoiceElementData is the API with which the voice element can get its
     * configuration and get/set variables. Returning null from the method
     * indicates the voice element is not done and Audium Call Services needs
     * to revisit it to produce the next VoiceXML page. Return a non-null exit
     * state to indicate the voice element is done.
     */
    protected String addXmlBody(VMain vxml, Hashtable reqParameters, VoiceElementData ved)
        throws VException, ElementException
    {
        // Get a reference to the preference object
        VPreference pref = ved.getPreference();

        // If we encountered max no match, exit as such
        String maxNoMatch = (String)reqParameters.get("maxNoMatch");
        if (maxNoMatch != null)
        {
            return "max_nomatch";
        }

        // If we encountered max no input, exit as such
        String maxNoInput = (String)reqParameters.get("maxNoInput");
        if (maxNoInput != null)
        {
            return "max_noinput";
        }

        // If we encountered confirm max no match, exit as such
        String maxConfirmNoMatch = (String)reqParameters.get("maxConfirmNoMatch");
        if (maxConfirmNoMatch != null)
        {
            return "max_confirm_nomatch";
        }

        // If we encountered confirm max no input, exit as such
        String maxConfirmNoInput = (String)reqParameters.get("maxConfirmNoInput");
        if (maxConfirmNoInput != null)
        {
            return "max_confirm_noinput";
        }

        // Try to find which phase this element was in last
        String lastPhase = (String)(ved.getScratchData("lastPhase"));
        if (lastPhase == null || lastPhase.equals("")) {

```

```

        lastPhase = "none";
    }

    // Create "start" form
    VForm startForm = VForm.getNew(pref, "start");

    // Phase 1:
    // *****
    if (lastPhase.equals("none") || lastPhase.equals("3"))
    {
        // 1. Generate school name list
        // The list of schools in the caller's area
        ArrayList schoolNames = new ArrayList();

        // Pull required variables out of the session
        String city = (String)ved.getSessionData("selectedCity");
        String state = (String)ved.getSessionData("selectedState");
        String schoolType = (String)ved.getSessionData("selectedSchoolType");

        // Find the list of schools
        schoolNames = findSchoolsIn(city, state, schoolType);

        // 2. Prompt user for a school name
        VoiceElementConfig vec = ved.getVoiceElementConfig();
        VoiceElementConfig.AudioGroup initialAG = vec.getAudioGroup(
            "initial_audio_group", 1);
        VMenuField field = VMenuField.getNew(pref, "schoolName", VMenuField.SPEECH);
        // Add each school name as an option in the field
        field.setPromptCount(1, initialAG.constructAudio(ved));
        for (int i = 0; i < schoolNames.size(); i++)
        {
            String schoolName = (String)schoolNames.get(i);
            VoiceInput vi = new VoiceInput(schoolName, schoolName, schoolName, false);
            field.setChoice(vi, null, null);
        }
        // Add the field to the form
        startForm.add(field);

        // 3. Add catch blocks for no match
        VEvent nomatchEvent = VEvent.getNew(pref, VEvent.NOMATCH);
        VAudio audioEvent = null;
        VAction logEvent = null;
        for (int i = 1; i <= vec.getIntSettingValue("max_nomatch_count", ved); i++) {
            VoiceElementConfig.AudioGroup audioGroup =
                ved.getVoiceElementConfig().getAudioGroup("nomatch_audio_group", i);

            if (audioGroup != null) {
                audioEvent = audioGroup.constructAudio(ved);
            }

            // Add the nomatch count to the log file
            logEvent = VAction.getNew(pref, VAction.ASSIGN,
                VXML_LOG_VARIABLE_NAME, VXML_LOG_VARIABLE_NAME + " + '|||'"
                + getElementName() + "_nomatch$$$" + "'" + i
                + "' + '^^^' + application.getElapsedTime("
                + ELEMENT_START_TIME_MILLISECS + ")",
                VAction.WITHOUT_QUOTES);

            nomatchEvent.addCount(i);
            nomatchEvent.addItem(i, audioEvent);
        }
    }

```



```

nomatchEvent.addItem(i, logEvent);

// Add audio played to the log file
logEvent = VAction.getNew(pref, VAction.ASSIGN,
    VXML_LOG_VARIABLE_NAME, VXML_LOG_VARIABLE_NAME
    + " + '|||audio_group$$$" + getElementName()
    + "_nomatch_audio_group"
    + "~~~~" + application.getElapsedTime("
    + ELEMENT_START_TIME_MILLISECS + ")",
    VAction.WITHOUT_QUOTES);

nomatchEvent.addItem(i, logEvent);
nomatchEvent.setReprompt(i, (audioEvent == null));
}
// Add submit to max no match event
VAction submitAction = VAction.getNew(pref, VAction.VARIABLE,
    "maxNoMatch", "yes", VAction.WITH_QUOTES);
submitAction.add(getSubmitURL(), VXML_LOG_VARIABLE_NAME + " maxNoMatch");
nomatchEvent.addItem(nomatchEvent.getMaxCount(), submitAction);

startForm.add(nomatchEvent);

// 4. Add catch blocks for no input
VEvent noinputEvent = VEvent.getNew(pref, VEvent.NOINPUT);
for (int i = 1; i <= vec.getIntSettingValue("max_noinput_count", ved); i++) {
    VoiceElementConfig.AudioGroup audioGroup =
        ved.getVoiceElementConfig().getAudioGroup("noinput_audio_group", i);

    if (audioGroup != null) {
        audioEvent = audioGroup.constructAudio(ved);
    }

    // Add the noinput count to the log file
    logEvent = VAction.getNew(pref, VAction.ASSIGN,
        VXML_LOG_VARIABLE_NAME, VXML_LOG_VARIABLE_NAME + " + '|||"
        + getElementName() + "_noinput$$$" + "'" + i
        + "' + '~~~~" + application.getElapsedTime("
        + ELEMENT_START_TIME_MILLISECS + ")",
        VAction.WITHOUT_QUOTES);

    noinputEvent.addCount(i);
    noinputEvent.addItem(i, audioEvent);
    noinputEvent.addItem(i, logEvent);

    // Add audio played to the log file
    logEvent = VAction.getNew(pref, VAction.ASSIGN,
        VXML_LOG_VARIABLE_NAME, VXML_LOG_VARIABLE_NAME
        + " + '|||audio_group$$$" + getElementName()
        + "_noinput_audio_group"
        + "~~~~" + application.getElapsedTime("
        + ELEMENT_START_TIME_MILLISECS + ")",
        VAction.WITHOUT_QUOTES);

    noinputEvent.addItem(i, logEvent);
    noinputEvent.setReprompt(i, (audioEvent == null));
}
// Add submit to max no input event
VAction submitAction2 = VAction.getNew(pref, VAction.VARIABLE,
    "maxNoInput", "yes", VAction.WITH_QUOTES);
submitAction2.add(getSubmitURL(), VXML_LOG_VARIABLE_NAME + " maxNoInput");

```



```

noinputEvent.addItem(noinputEvent.getMaxCount(), submitAction2);

startForm.add(noinputEvent);

// Submit the input back
VAction submitAction3 = getSubmitVAction("schoolName", pref);
submitAction3.setEncoding("multipart/form-data");
startForm.add(submitAction3);

// Store that we completed phase 1 in the scratch data
ved.setScratchData("lastPhase", "1");

// Return null because the element is not done yet
vxml.add(startForm); // Add the form to the VoiceXML
return null;
}

// Phase 2:
// *****
else if (lastPhase.equals("1"))
{
    // 1. Prompt user for confirmation of school name
    // Get the school name
    String schoolName = (String)reqParameters.get("schoolName");
    // Set it in session data
    try
    {
        ved.setSessionData("selectedSchoolName", schoolName);
    }
    catch (AudiumException ae)
    {
        ae.printStackTrace();
    }
    // Create and add the prompt to the form
    VoiceElementConfig vec = ved.getVoiceElementConfig();
    VoiceElementConfig.AudioGroup confirmAG = vec.getAudioGroup(
        "Confirm", 1);
    VBuiltInField field = VBuiltInField.getNew(
        pref, VBuiltInField.BOOLEAN, "confirm",
        VBuiltInField.DTMF_SPEECH);
    field.setPromptCount(1, confirmAG.constructAudio(ved));
    // Add the field to the form
    startForm.add(field);

    // 2. Add catch block for no match during confirmation
    VEvent nomatchEvent = VEvent.getNew(pref, VEvent.NOMATCH);
    VAudio audioEvent = null;
    VAction logEvent = null;
    for (int i = 1; i <= vec.getIntSettingValue("max_confirm_nomatch_count", ved);
i++) {
        VoiceElementConfig.AudioGroup audioGroup =
            ved.getVoiceElementConfig().getAudioGroup("Confirm_NoMatch", i);

        if (audioGroup != null) {
            audioEvent = audioGroup.constructAudio(ved);
        }

        // Add the nomatch count to the log file
        logEvent = VAction.getNew(pref, VAction.ASSIGN,
            VXML_LOG_VARIABLE_NAME, VXML_LOG_VARIABLE_NAME + " + '|||'"

```

```

        + getElementName() + "_nomatch$$$" + "'" + i
        + "' + '^~^' + application.getElapsedTime("
        + ELEMENT_START_TIME_MILLISECS + ")",
        VAction.WITHOUT_QUOTES);

nomatchEvent.addCount(i);
nomatchEvent.addItem(i, audioEvent);
nomatchEvent.addItem(i, logEvent);

// Add audio played to the log file
logEvent = VAction.getNew(pref, VAction.ASSIGN,
    VXML_LOG_VARIABLE_NAME, VXML_LOG_VARIABLE_NAME
    + " + '|||audio_group$$$" + getElementName()
    + "_confirm_nomatch_audio_group"
    + "^~^" + application.getElapsedTime("
    + ELEMENT_START_TIME_MILLISECS + ")",
    VAction.WITHOUT_QUOTES);

nomatchEvent.addItem(i, logEvent);
nomatchEvent.setReprompt(i, (audioEvent == null));
}

// Add submit to max no match event
VAction submitAction = VAction.getNew(pref, VAction.VARIABLE,
    "maxConfirmNoMatch", "yes", VAction.WITH_QUOTES);
submitAction.add(getSubmitURL(), VXML_LOG_VARIABLE_NAME + " maxConfirmNoMatch");
nomatchEvent.addItem(nomatchEvent.getMaxCount(), submitAction);

startForm.add(nomatchEvent);

// 3. Add catch block for no input during confirmation
VEvent noinputEvent = VEvent.getNew(pref, VEvent.NOINPUT);
for (int i = 1; i <= vec.getIntSettingValue("max_confirm_noinput_count", ved);
i++) {
    VoiceElementConfig.AudioGroup audioGroup =
        ved.getVoiceElementConfig().getAudioGroup("Confirm_NoInput", i);

    if (audioGroup != null) {
        audioEvent = audioGroup.constructAudio(ved);
    }

    // Add the noinput count to the log file
    logEvent = VAction.getNew(pref, VAction.ASSIGN,
        VXML_LOG_VARIABLE_NAME, VXML_LOG_VARIABLE_NAME + " + '|||"
        + getElementName() + "_noinput$$$" + "'" + i
        + "' + '^~^' + application.getElapsedTime("
        + ELEMENT_START_TIME_MILLISECS + ")",
        VAction.WITHOUT_QUOTES);

    noinputEvent.addCount(i);
    noinputEvent.addItem(i, audioEvent);
    noinputEvent.addItem(i, logEvent);

    // Add audio played to the log file
    logEvent = VAction.getNew(pref, VAction.ASSIGN,
        VXML_LOG_VARIABLE_NAME, VXML_LOG_VARIABLE_NAME
        + " + '|||audio_group$$$" + getElementName()
        + "_confirm_noinput_audio_group"
        + "^~^" + application.getElapsedTime("
        + ELEMENT_START_TIME_MILLISECS + ")",

```

```

        VAction.WITHOUT_QUOTES);

        noinputEvent.addItem(i, logEvent);
        noinputEvent.setReprompt(i, (audioEvent == null));
    }
    // Add submit to max no input event
    VAction submitAction2 = VAction.getNew(pref, VAction.VARIABLE,
        "maxConfirmNoInput", "yes", VAction.WITH_QUOTES);
    submitAction2.add(getSubmitURL(), VXML_LOG_VARIABLE_NAME + " maxConfirmNoInput");
    noinputEvent.addItem(noinputEvent.getMaxCount(), submitAction2);

    startForm.add(noinputEvent);

    // Submit the what the caller said (yes/no)
    VAction submitAction3 = getSubmitVAction("", pref);
    submitAction3.setEncoding("multipart/form-data");
    startForm.add(submitAction3);

    // Store that we completed phase 2 in the scratch data
    ved.setScratchData("lastPhase", "2");

    // Return null because the element is not done yet
    vxml.add(startForm); // Add the form to the VoiceXML
    return null;
}

// Phase 3:
// *****
else if (lastPhase.equals("2"))
{
    // Find caller input
    String confirm = (String)reqParameters.get("confirm");

    // 1. If caller confirmed, exit as "done"
    if (confirm.equalsIgnoreCase("true") ||
        confirm.equalsIgnoreCase("1"))
    {
        return "done";
    }
    else
    {
        // 2. If caller disconfirmed, increment disconfirm counter.
        // If counter > max_disconfirm_count, exit as "max_disconfirm".
        // Otherwise, go back to first pass.
        String disconfirmCount = (String)ved.getScratchData("disconfirmCount");
        if (disconfirmCount == null || disconfirmCount.equals(""))
        {
            disconfirmCount = "0";
        }
        disconfirmCount = "" + (Integer.parseInt(disconfirmCount) + 1);
        ved.setScratchData("disconfirmCount", disconfirmCount);
        VoiceElementConfig vec = ved.getVoiceElementConfig();
        int maxDisconfirmCount = vec.getIntSettingValue("max_disconfirm_count", ved);
        if (Integer.parseInt(disconfirmCount) >= maxDisconfirmCount)
        {
            return "max_disconfirmed";
        }
    }

    // Store that we completed phase 3 in the scratch data
    ved.setScratchData("lastPhase", "3");
}

```

```

        // Go back to phase 1 (call self, will go to phase 1)
        return this.addXmlBody(vxml, reqParameters, ved);
    }
}

else
{
    // catch-all (should never get here)
    return "done";
}
}

// Returns the given ArrayList as a String array
// NOTE: The ArrayList is assumed to contain only strings
private String[] arrayListToStringArray(ArrayList al)
{
    String stringArray[] = new String[al.size()];
    for (int i = 0; i < al.size(); i++)
    {
        stringArray[i] = (String)(al.get(i));
    }

    return stringArray;
}

// Returns the NPA (i.e. area code) of the given ANI
private String getUserNPA(String userANI)
{
    String NPA = userANI.substring(0, 2);
    return NPA;
}

// Returns an ArrayList of schools in the city and state, and
// of the requested type
private ArrayList findSchoolsIn(String city, String state, String schoolType)
{
    // Instantiate a database access object
    DBFunctionalityClass dbObj = new DBFunctionalityClass();

    // The ArrayList of school names
    ArrayList schoolNames = new ArrayList();

    try
    {
        String strSQL = "";
        strSQL = "SELECT DISTINCT SchoolName, SchoolType FROM schools_tb WHERE state = '"
+ state + "' AND SchoolType = '" + schoolType + "' AND city = '" + city + "'";
        ResultSet rs = dbObj.performQuery(strSQL);
        if(rs != null)
        {
            boolean more = rs.next();
            while(more)
            {
                String sName = rs.getString("SchoolName");
                schoolNames.add(sName);
                more = rs.next();
            }
            rs.close();
        }
    }
}

```

```

    }
    catch(SQLException e)
    {
        e.printStackTrace();
    }

    return schoolNames;
}

private ArrayList findSchoolsIn(String NPA, String schoolType)
{
    // Instantiate a database access object
    DBFunctionalityClass dbObj = new DBFunctionalityClass();

    // The ArrayList of school names
    ArrayList schoolNames = new ArrayList();

    try
    {
        String strSQL = "";
        // TODO: Change this query to use state, city, and school type instead of NPA
        strSQL = "SELECT DISTINCT SchoolName, SchoolType FROM schools_tb WHERE
substring(PhoneNumber from 1 for 3) = '" + NPA + "' AND SchoolType = '" + schoolType + "'";
        ResultSet rs = dbObj.performQuery(strSQL);
        if(rs != null)
        {
            boolean more = rs.next();
            while(more)
            {
                String sName = rs.getString("SchoolName");
                schoolNames.add(sName);
                more = rs.next();
            }
            rs.close();
        }
    }
    catch(SQLException e)
    {
        e.printStackTrace();
    }
    return schoolNames;
}

/**
 * This method returns the name the voice element will have in the Element
 * Pane in the Audium Builder for Studio.
 */
public String getElementName()
{
    return "School_Name_With_Confirm";
}

/**
 * This method returns the name of the folder in which this voice element
 * resides. Return null if it is to appear directly under the Elements
 * folder.
 */
public String getDisplayFolderName()
{
    return "ITS0";
}

```

```

    }

    /**
     * This method returns the text of a description of the voice element that
     * will appear as a popup when the cursor points to the element.
     */
    public String getDescription()
    {
        return "Captures the name of a school in the caller's area (based on ANI).";
    }

    /**
     * This method returns an array of Setting objects representing all the
     * settings this voice element expects. Return null if the voice element
     * does not need any settings.
     */
    public Setting[] getSettings() throws ElementException
    {
        Setting[] settingArray = new Setting[5];

        settingArray[0] = new Setting(Setting.MAX_NOINPUT_COUNT);
        settingArray[1] = new Setting(Setting.MAX_NOMATCH_COUNT);
        settingArray[2] = new Setting("max_confirm_noinput_count",
            "Max Confirm NoInput Count",
            "The number of noinput events during confirmation after which max noinput
occurs.",
            true, true, true, Setting.INT);
        settingArray[3] = new Setting("max_confirm_nomatch_count",
            "Max Confirm NoMatch Count",
            "The number of nomatch events during confirmation after which max nomatch
occurs.",
            true, true, true, Setting.INT);
        settingArray[4] = new Setting("max_disconfirm_count",
            "Max Disconfirm Count",
            "The maximum number of times the caller can disconfirm.",
            true, true, true, Setting.INT);

        return settingArray;
    }

    /**
     * This method returns a HashMap of arrays of AudioGroup objects
     * representing each set of audio groups this voice element expects. The
     * keys to the HashMap are the names of sets of audio groups. The arrays
     * represent the audio groups that appear in the set. Return null if the
     * voice element does not need any audio groups.
     */
    public HashMap getAudioGroups() throws ElementException
    {
        HashMap groups = new HashMap();

        // Audio group for school name
        AudioGroup[] audioGroupArray1 = new AudioGroup[3];
        audioGroupArray1[0] = new AudioGroup(AudioGroup.INITIAL);
        audioGroupArray1[1] = new AudioGroup(AudioGroup.NOMATCH);
        audioGroupArray1[2] = new AudioGroup(AudioGroup.NOINPUT);

        groups.put("School Capture", audioGroupArray1);

        // Audio group for confirmation
    }

```

```

AudioGroup[] audioGroupArray2 = new AudioGroup[3];
audioGroupArray2[0] = new AudioGroup("Confirm",
    "Confirm", "Confirmation prompt", true, true);
audioGroupArray2[1] = new AudioGroup("Confirm_NoInput",
    "Confirm_NoInput", "Confirmation no input", false, false);
audioGroupArray2[2] = new AudioGroup("Confirm_NoMatch",
    "Confirm_NoMatch", "Confirmation no match", false, false);

groups.put("School Confirm", audioGroupArray2);

return groups;
}

/**
 * This method returns an array of Strings representing the order in which
 * the audio group sets are to be displayed in Audium Builder for Studio.
 * Return null if the voice element does not need any audio groups.
 */
public String[] getAudioGroupDisplayOrder()
{
    String[] displayOrder = new String[2];

    displayOrder[0] = "School Capture";
    displayOrder[1] = "School Confirm";

    return displayOrder;
}

/**
 * This method returns an array of ExitState objects representing all the
 * possible exit states the voice element can return. There must be at
 * least one exit state.
 */
public ExitState[] getExitStates() throws ElementException
{
    ExitState exitStates[] = new ExitState[6];

    // Done
    ExitState done = new ExitState(ExitState.DONE);
    exitStates[0] = done;
    // Max No Input
    ExitState max_noinput = new ExitState(ExitState.MAX_NOINPUT);
    exitStates[1] = max_noinput;
    // Max No Match
    ExitState max_nomatch = new ExitState(ExitState.MAX_NOMATCH);
    exitStates[2] = max_nomatch;
    // Max Disconfirmed
    ExitState max_disconfirmed = new ExitState(ExitState.MAX_DISCONFIRMED);
    exitStates[3] = max_disconfirmed;
    // Max Confirm No Match
    ExitState max_confirm_nomatch =
        new ExitState("max_confirm_nomatch", "max_confirm_nomatch",
            "The caller has confirmed improperly too many times.");
    exitStates[4] = max_confirm_nomatch;
    // Max Confirm No Input
    ExitState max_confirm_noinput =
        new ExitState("max_confirm_noinput", "max_confirm_noinput",
            "The caller has failed to provide confirmation input too many times.");
    exitStates[5] = max_confirm_noinput;
}

```



```

        return exitStates;
    }

    /**
     * This method returns an array of ElementData objects representing the
     * element data that this voice element creates. Return null if the voice
     * element does not create any Element Data.
     */
    public ElementData[] getElementData() throws ElementException
    {
        return null;
    }
}

```

11. We now know: the state, city, school type, and school name. These values have all been stored in session variables. All that remains to be done is to query the database and obtain the information about the relevant school, then read it back to the user. First we query the database using a custom action element to obtain the unique identifier for the school. Create an action element called `getSchoolID_action` and the java class `GetSchoolIDActionClass` as shown in Figure 4-198 on page 298.

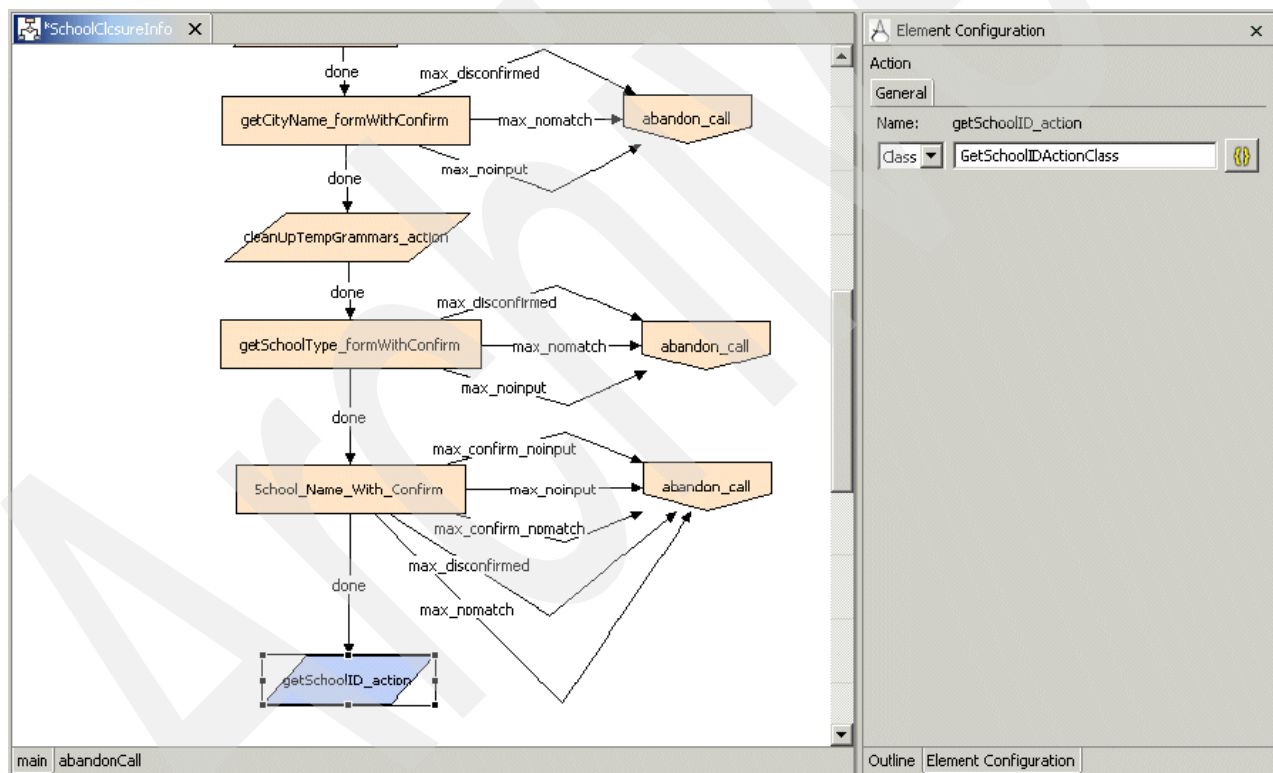


Figure 4-198 SchoolClosureInfo: getSchoolID\_action element

The `GetSchoolIDActionClass` will have the source code shown in Example 4-22.

#### Example 4-22 GetSchoolIDActionClass

```

/*

Given a school name, state, school type and city we now retrieve the school id.

*/

```

```

import com.audium.server.AudiumException;
import com.audium.server.session.ActionElementData;
import com.audium.server.voiceElement.ActionElementBase;
import java.sql.*;

public class GetSchoolIDActionClass extends ActionElementBase
{
    public void doAction(String name, ActionElementData actionData) throws AudiumException
    {
        String city, state, schoolName, schoolType = "";
        city = actionData.getSessionData("selectedCity").toString();
        state = getStateAbbreviation(actionData.getSessionData("selectedState").toString());
        schoolName = actionData.getSessionData("selectedSchoolName").toString();
        schoolType = actionData.getSessionData("selectedSchoolType").toString();
        DBFunctionalityClass dbObj = new DBFunctionalityClass();
        try
        {
            String strSQL = "SELECT id FROM schools_tb WHERE SchoolName = '" + schoolName +
                "' AND City = '" + city + "' AND SchoolType = '" + schoolType + "' AND State = '" + state
                + "' LIMIT 1";
            ResultSet rs = dbObj.performQuery(strSQL);
            if(rs != null)
            {
                boolean more = rs.next();
                while(more)
                {
                    String schoolId = rs.getString("id");
                    actionData.setSessionData("selectedSchoolId", schoolId);
                    more = rs.next();
                }
                rs.close();
            }
        }
        catch(SQLException e)
        {
            e.printStackTrace();
            actionData.setSessionData("selectedSchoolId", "");
        }
    }

    private String getStateAbbreviation(String fullStateName)
    {
        //this method takes a full state name as a param and return the abbreviation
        String abbreviation = "";
        if(fullStateName.equalsIgnoreCase("ALABAMA"))
        {
            abbreviation = "AL";
        }
        else if(fullStateName.equalsIgnoreCase("ALASKA"))
        {
            abbreviation = "AK";
        }
        else if(fullStateName.equalsIgnoreCase("ARIZONA"))
        {
            abbreviation = "AZ";
        }
        else if(fullStateName.equalsIgnoreCase("ARKANSAS"))
        {
            abbreviation = "AR";
        }
    }
}

```

```

else if(fullStateName.equalsIgnoreCase("CALIFORNIA"))
{
    abbreviation = "CA";
}
else if(fullStateName.equalsIgnoreCase("COLORADO"))
{
    abbreviation = "CO";
}
else if(fullStateName.equalsIgnoreCase("CONNECTICUT"))
{
    abbreviation = "CT";
}
else if(fullStateName.equalsIgnoreCase("DELAWARE"))
{
    abbreviation = "DE";
}
else if(fullStateName.equalsIgnoreCase("DISTRICT OF COLUMBIA") ||
fullStateName.equalsIgnoreCase("D C"))
{
    abbreviation = "DC";
}
else if(fullStateName.equalsIgnoreCase("FLORIDA"))
{
    abbreviation = "FL";
}
else if(fullStateName.equalsIgnoreCase("GEORGIA"))
{
    abbreviation = "GA";
}
else if(fullStateName.equalsIgnoreCase("HAWAII"))
{
    abbreviation = "HI";
}
else if(fullStateName.equalsIgnoreCase("IDAHO"))
{
    abbreviation = "ID";
}
else if(fullStateName.equalsIgnoreCase("ILLINOIS"))
{
    abbreviation = "IL";
}
else if(fullStateName.equalsIgnoreCase("INDIANA"))
{
    abbreviation = "IN";
}
else if(fullStateName.equalsIgnoreCase("IOWA"))
{
    abbreviation = "IA";
}
else if(fullStateName.equalsIgnoreCase("KANSAS"))
{
    abbreviation = "KS";
}
else if(fullStateName.equalsIgnoreCase("KENTUCKY"))
{
    abbreviation = "KY";
}
else if(fullStateName.equalsIgnoreCase("LOUISIANA"))
{
    abbreviation = "LA";
}

```

```

}
else if(fullStateName.equalsIgnoreCase("MAINE"))
{
    abbreviation = "ME";
}
else if(fullStateName.equalsIgnoreCase("MARYLAND"))
{
    abbreviation = "MD";
}
else if(fullStateName.equalsIgnoreCase("MASSACHUSETTS"))
{
    abbreviation = "MA";
}
else if(fullStateName.equalsIgnoreCase("MICHIGAN"))
{
    abbreviation = "MI";
}
else if(fullStateName.equalsIgnoreCase("MINNESOTA"))
{
    abbreviation = "MN";
}
else if(fullStateName.equalsIgnoreCase("MISSISSIPPI"))
{
    abbreviation = "MS";
}
else if(fullStateName.equalsIgnoreCase("MISSOURI"))
{
    abbreviation = "MO";
}
else if(fullStateName.equalsIgnoreCase("MONTANA"))
{
    abbreviation = "MT";
}
else if(fullStateName.equalsIgnoreCase("NEBRASKA"))
{
    abbreviation = "NE";
}
else if(fullStateName.equalsIgnoreCase("NEVADA"))
{
    abbreviation = "NV";
}
else if(fullStateName.equalsIgnoreCase("NEW HAMPSHIRE"))
{
    abbreviation = "NH";
}
else if(fullStateName.equalsIgnoreCase("NEW JERSEY"))
{
    abbreviation = "NJ";
}
else if(fullStateName.equalsIgnoreCase("NEW MEXICO"))
{
    abbreviation = "NM";
}
else if(fullStateName.equalsIgnoreCase("NEW YORK"))
{
    abbreviation = "NY";
}
else if(fullStateName.equalsIgnoreCase("NORTH CAROLINA"))
{
    abbreviation = "NC";
}

```

```

}
else if(fullStateName.equalsIgnoreCase("NORTH DAKOTA"))
{
    abbreviation = "ND";
}
else if(fullStateName.equalsIgnoreCase("OHIO"))
{
    abbreviation = "OH";
}
else if(fullStateName.equalsIgnoreCase("OKLAHOMA"))
{
    abbreviation = "OK";
}
else if(fullStateName.equalsIgnoreCase("OREGON"))
{
    abbreviation = "OR";
}
else if(fullStateName.equalsIgnoreCase("PENNSYLVANIA"))
{
    abbreviation = "PA";
}
else if(fullStateName.equalsIgnoreCase("RHODE ISLAND"))
{
    abbreviation = "RI";
}
else if(fullStateName.equalsIgnoreCase("SOUTH CAROLINA"))
{
    abbreviation = "SC";
}
else if(fullStateName.equalsIgnoreCase("SOUTH DAKOTA"))
{
    abbreviation = "SD";
}
else if(fullStateName.equalsIgnoreCase("TENNESSEE"))
{
    abbreviation = "TN";
}
else if(fullStateName.equalsIgnoreCase("TEXAS"))
{
    abbreviation = "TX";
}
else if(fullStateName.equalsIgnoreCase("UTAH"))
{
    abbreviation = "UT";
}
else if(fullStateName.equalsIgnoreCase("VERMONT"))
{
    abbreviation = "VT";
}
else if(fullStateName.equalsIgnoreCase("VIRGINIA"))
{
    abbreviation = "VA";
}
else if(fullStateName.equalsIgnoreCase("WASHINGTON"))
{
    abbreviation = "WA";
}
else if(fullStateName.equalsIgnoreCase("WEST VIRGINIA"))
{
    abbreviation = "WV";
}

```

```

    }
    else if(fullStateName.equalsIgnoreCase("WISCONSIN"))
    {
        abbreviation = "WI";
    }
    else if(fullStateName.equalsIgnoreCase("WYOMING"))
    {
        abbreviation = "WY";
    }
    return abbreviation;
}
}

```

12. Now, that we have the SchoolId we can query the database for the information that we want to read back to the user. Note that we could have obtained the SchoolId and school information in the same step, but we have chosen to separate these two actions. Why? We have done this to remind you that obtaining the SchoolId is an essential step in determining whether or not we have enough information to identify a unique, single school. If our query returns only one row, then we have narrowed our selection down to one school successfully, otherwise further refinements to the search will be required to narrow our selection down to a unique school. Again, we leave the implementation of this functionality as an programming exercise for you. Create the getSchoolInfoByID\_action element and the GetSchoolInfoByIdActionClass. Refer to Figure 4-199.

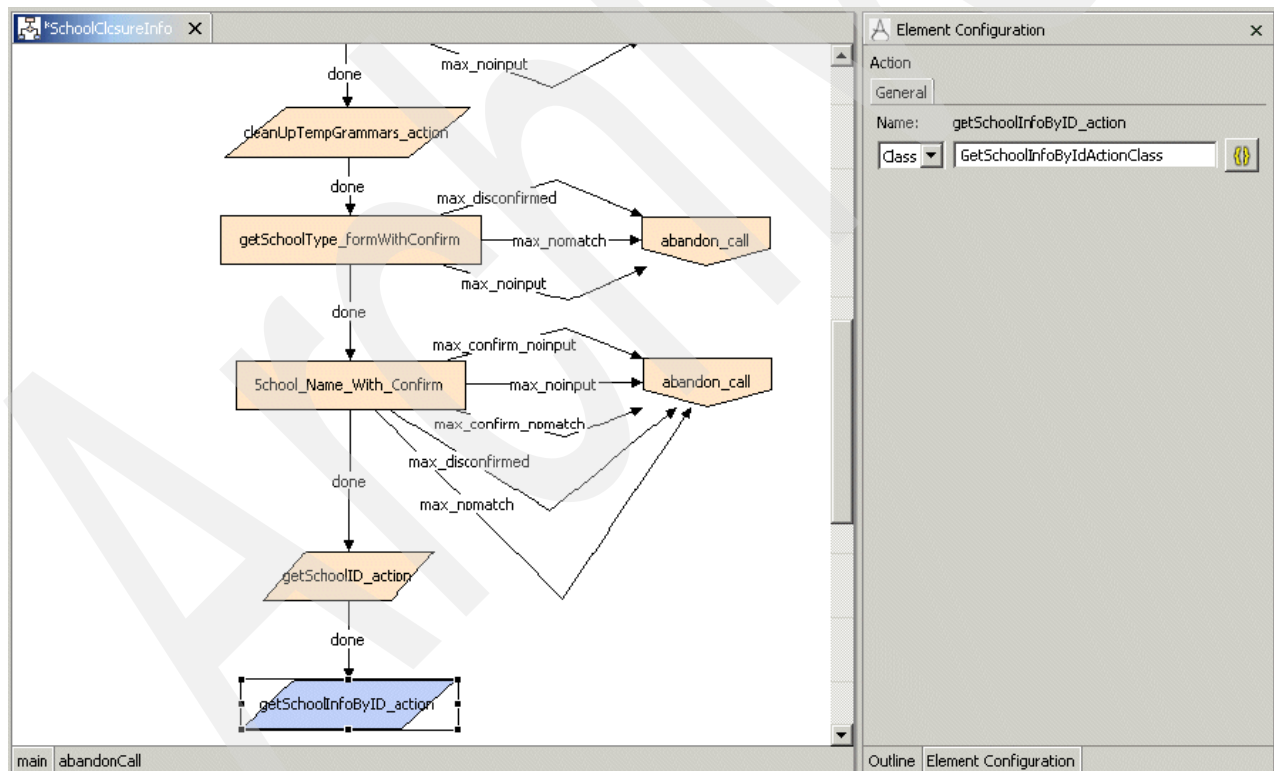


Figure 4-199 SchoolClosureInfo: getSchoolInfoByID\_action element

GetSchoolInfoByIdActionClass will have the source code shown in Example 4-23 on page 304.

```

/*

Given the desired school name (taken either from the "schoolName"
session variable or the value field of the "getSchoolName_form"
this class will query the database to obtain the information
required for read back to the user.

*/

import com.audium.server.AudiumException;
import com.audium.server.session.ActionElementData;
import com.audium.server.voiceElement.ActionElementBase;
import java.sql.*;
//import java.io.*;
//import java.util.*;

public class GetSchoolInfoByIdActionClass extends ActionElementBase
{
    public void doAction(String name, ActionElementData actionData) throws AudiumException
    {
        String schoolId = actionData.getSessionData("selectedSchoolId").toString();
        try
        {
            DBFunctionalityClass dbObj = new DBFunctionalityClass();
            String strSQL = "SELECT * FROM schools_tb WHERE id = '" + schoolId + "'";
            ResultSet rs = dbObj.performQuery(strSQL);
            if(rs != null)
            {
                boolean more = rs.next();
                while(more)
                {
                    actionData.setSessionData("selectedSchoolName",
rs.getString("SchoolName"));
                    actionData.setSessionData("schoolStreet", rs.getString("Street"));
                    actionData.setSessionData("schoolCity", rs.getString("City"));
                    actionData.setSessionData("schoolState", rs.getString("State"));
                    actionData.setSessionData("schoolZip", rs.getString("Zip"));
                    more = rs.next();
                }
                rs.close();

                strSQL = "SELECT Description FROM status_codes_tb WHERE id = (SELECT
CurrentStatusCode FROM current_status_tb WHERE SchoolId = " + schoolId + " AND (SinceDate
<= now() AND UntilDate >= now()))";
                rs = dbObj.performQuery(strSQL);
                if(rs != null)
                {
                    boolean more = rs.next();
                    while(more)
                    {
                        actionData.setSessionData("currentSchoolStatus",
rs.getString("Description"));
                        more = rs.next();
                    }
                    rs.close();
                }

                strSQL = "SELECT untilDate, untilDateDefinite, additionalInfoTTS FROM
current_status_tb WHERE schoolID = '" + schoolId + "' AND (SinceDate <= now() AND
UntilDate >= now())";

```



```

rs = dbObj.performQuery(strSQL);
if(rs != null)
{
    boolean more = rs.next();
    while(more)
    {
        actionData.setSessionData("anticipatedStatusChangeDate",
rs.getString("untilDate").replaceAll("-", ""));
        actionData.setSessionData("additionalInfo",
rs.getString("additionalInfoTTS"));
        String changeType = rs.getString("untilDateDefinite");
        if(changeType.equalsIgnoreCase("YES"))
        {
            actionData.setSessionData("statusChangeDateDefiniteOrTentative",
"definite");
        }
        else
        {
            actionData.setSessionData("statusChangeDateDefiniteOrTentative",
"tentative");
        }
        more = rs.next();
    }
    rs.close();
}
}
catch(SQLException e)
{
    e.printStackTrace();
}
}
}

```

---

13. We have now retrieved all the information we need to read back to the user and stored it in logically named session variables. We read back this information using an audio element that simply accesses the information stored in session. To make things a little more elegant, we use a Say It Smart plug-in for this audio element to read back specially formatted information. Please refer to Figure 4-200 on page 306.

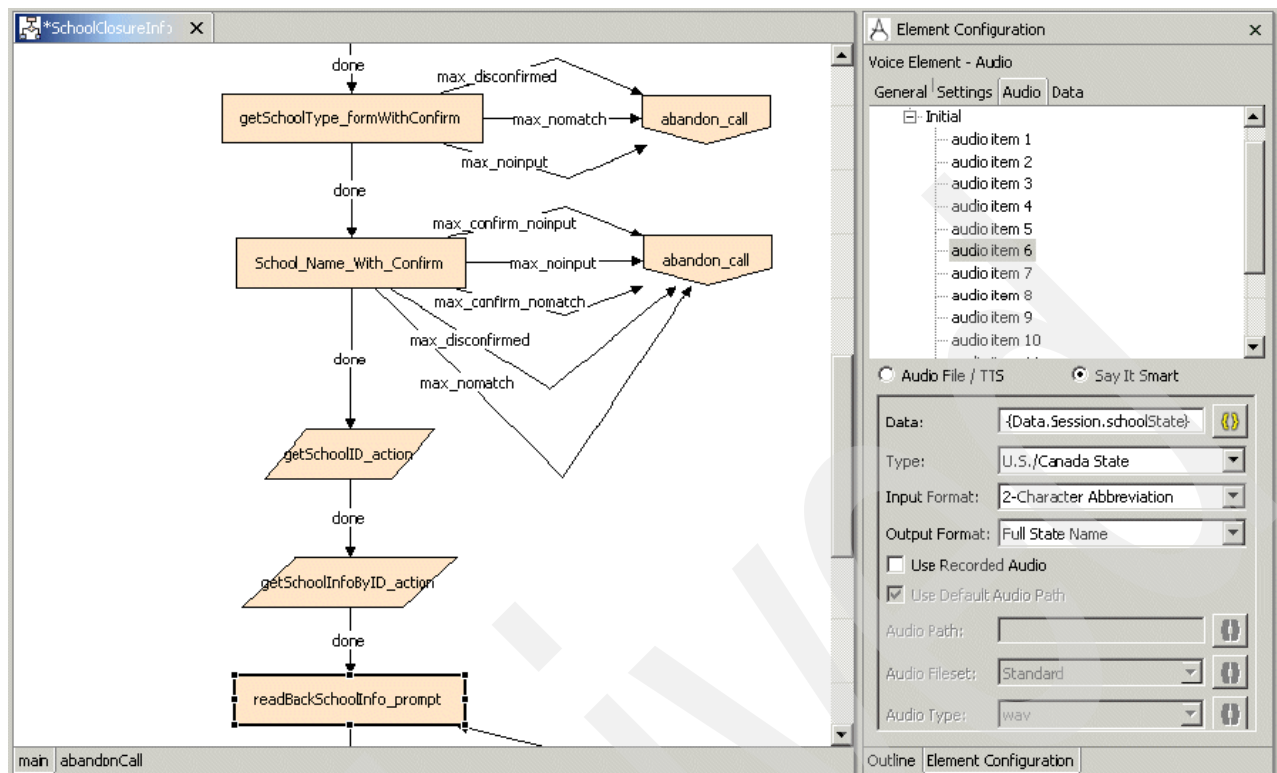


Figure 4-200 SchoolClosureInfo: Say It Smart plug-in

Table 4-17 outlines the settings for each individual audio item.

Table 4-17 Audio item settings

Audio item name	Audio File/TTS or Say It Smart	TTS Content	Say It Smart Config
Audio item 1		Thank you.	
Audio item 2		{Data.Session.selectedSchoolName}	
Audio item 3		located at	
Audio item 4		{Data.Session.school Street}	
Audio item 5		{Data.Session.school City}	
Audio item 6			Data: {Data.Session.schoolState} Type: US/Canada State Input Format: 2-Character Abbreviation Output Format: Full State Name

Audio item name	Audio File/TTS or Say It Smart	TTS Content	Say It Smart Config
Audio item 7			Data: {Data.Session.schoolZip} Type: Digit-By-Digit Input Format: Any length number Output Format: Digit-By-Digit
Audio item 8		is currently	
Audio item 9		{Data.Session.currentSchoolStatus}	
Audio item 10		until	
Audio item 11			Data: {Data.Session.anticipatedStatusChangeDate} Type: Date Input Format: YYYYMMDD Output Format: The Date
Audio item 12		this date is	
Audio item 13		{Data.Session.statusChangeDateDefiniteOrTentative}	
Audio item 14		The school's administration has provided the following additional information:	
Audio item 15			{Data.Session.additionalInfo}

Given this type of configuration, we could expect a typical information read back to go as follows:

“Thank you. Gardner Elementary School located at: 2134 Stormsurge Road Palookaville Sout Dakota, 9 0 6 7 8 is currently closed due to inclement weather until Thursday January 5<sup>th</sup>, 2006. This date is tentative. The school’s administration has provided the following additional information: please call back on a daily basis to obtain a status update. We anticipate reopening of the school as weather conditions improve. Thank you.”

14. We ask the user if he or she would like to hear the information again using a simple two-option menu element. Please refer to Figure 4-201.

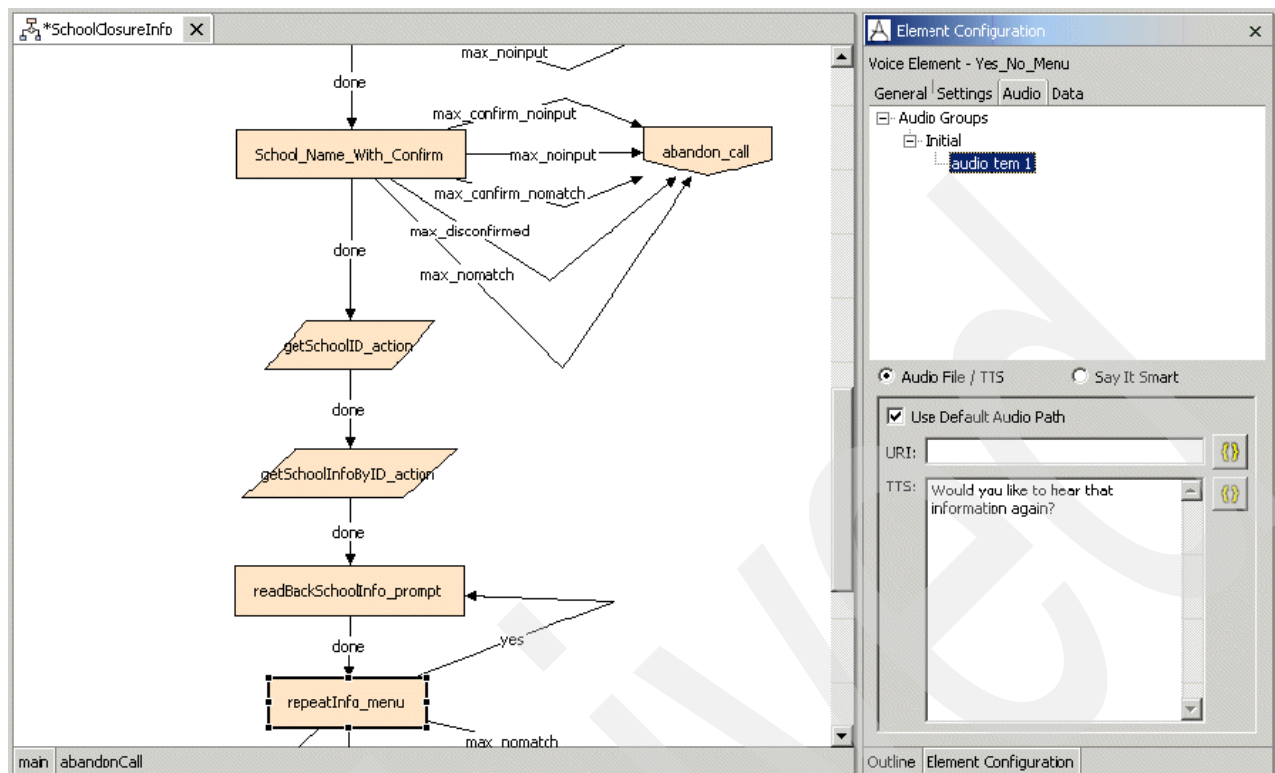


Figure 4-201 SchoolClosureInfo: repeatInfo\_menu

15. Ask the user if they would like to obtain information about another school. If yes, simply drop the caller back at the sayStateName\_audio element to restart the process. Please refer to Figure 4-202 on page 309.
16. Finally, if the user does not want any further information, we gracefully exit the application with a polite goodbye prompt. Please refer to Figure 4-203 on page 309.

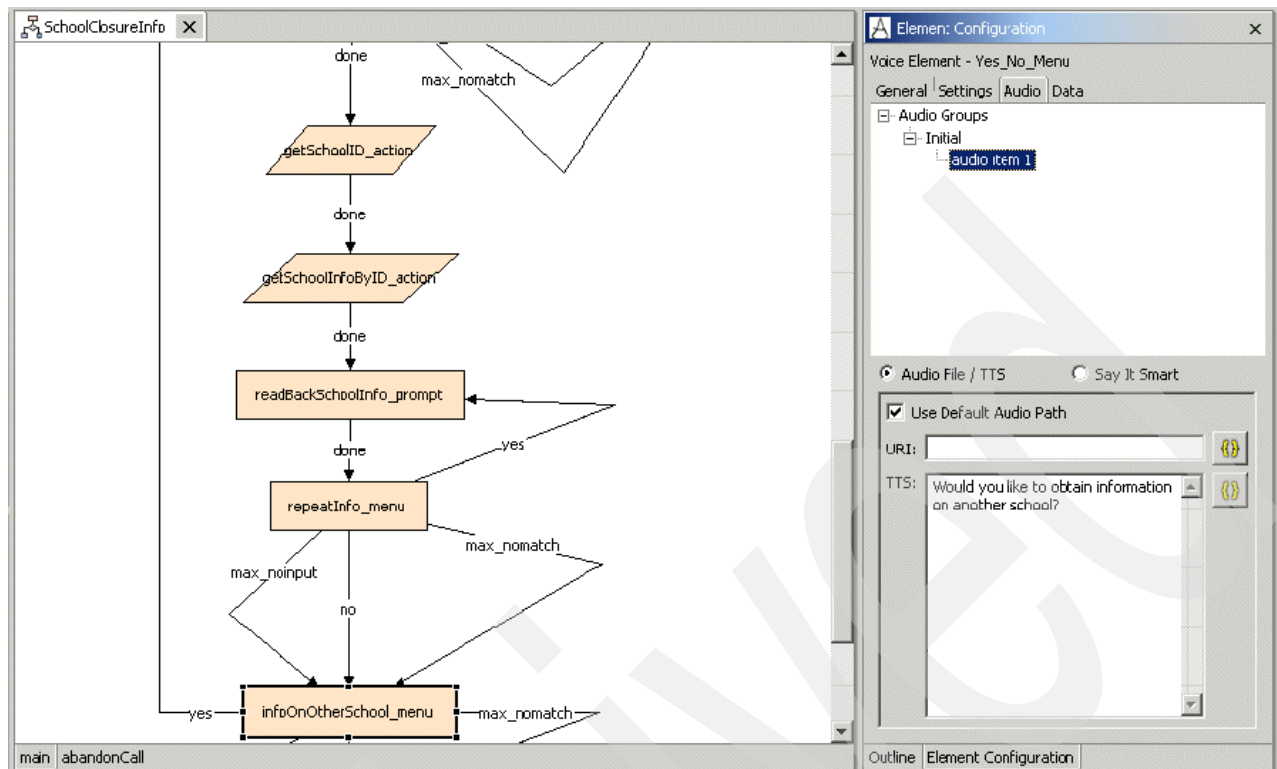


Figure 4-202 SchoolClosureInfo: InfoOnOtherSchool\_menu

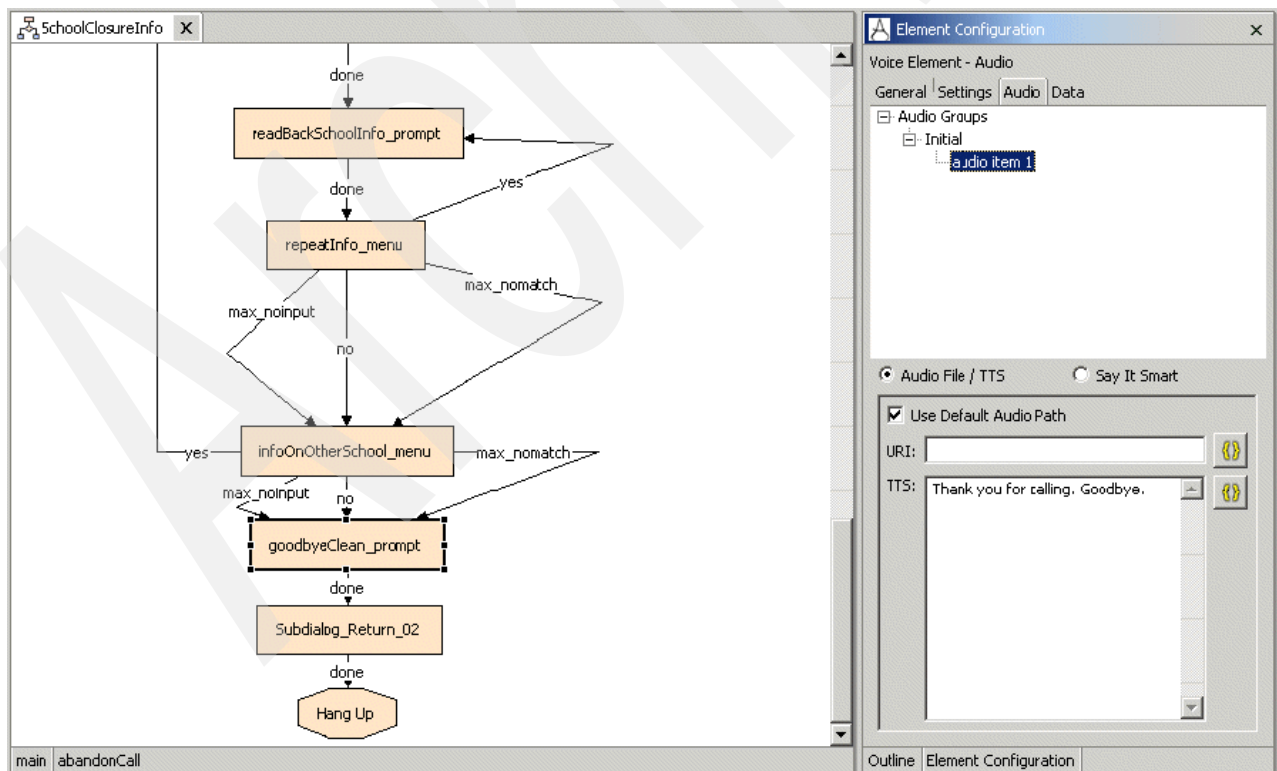


Figure 4-203 SchoolClosureInfo: goodbyeClean\_prompt

## 4.6 Integrating an application with ICM call routing

In this section, we demonstrate how a Cisco CVP V3.1 VoiceXML Server application can be integrated with an ICM routing script. As a by-product, we also show one way of developing multilingual applications.

The general call flow of an ICM Integrated application is of necessity somewhat different than that of a Standalone application. In Standalone mode, calls arrive at the gateway, and are handed directly to the appropriate CVP VoiceXML Server application for processing. When the application ends, the call is done.

In ICM Integrated mode, calls which arrive at the Ingress Gateway are handed to an ICM routing script first. The ICM routing script then transfers the call to the VoiceXML Gateway, and instructs the VoiceXML Gateway to invoke the CVP VoiceXML Server application. When the application ends, it passes data values back to the ICM routing script, which then continues its operation. Thus, the VoiceXML Server operates as a slave to the ICM, executing applications as requested by the ICM routing script.

### 4.6.1 Example overview

This is primarily a self-service application, but it is followed by an optional transfer to agent. The self-service application logic is defined in VoiceXML Studio, and the agent selection and transfer logic is defined in the ICM.

A call arrives in the system. The caller is welcomed to the California Chamber of Commerce Weather Report line, and asked to press 1 for English, 2 for French. Then, in his chosen language, he hears a recorded announcement describing the weather in California. Then the caller is given the opportunity to transfer to a travel agent. If he chooses to do so, he is transferred to either the English speaking agent or the French speaking agent, depending on his earlier language selection.

This application demonstrates a simple mechanism for implementing a multilingual capability. For user input, only DTMF is used, so multilingualism is not required. For output, recorded .wav files are used. No text-to-speech is involved in this application.

The application also demonstrates how data can be returned to ICM and acted upon there. We see that the caller's language choice is returned in a variable to the ICM routing script, which then transfers the call to one of two agents based on the contents of that variable.

In the sections that follow, we create the VoiceXML Studio application and perform the configuration necessary to test it in Standalone mode. Finally we create the ICM routing script, and perform the configuration necessary to invoke that routing script.

### 4.6.2 WeatherReport VoiceXML Studio application

First we create a VoiceXML Studio application called WeatherReport. Since we are not using ASR or TTS, we will select the DTMF-Only adapter. We are using recorded prompts, so we need to set the Default Audio Path to /CVP/audio/WeatherReport/, as shown in Figure 4-204 on page 311. We place our recorded audio files, then, on the VoiceXML Server machine in a directory called C:\Cisco\CVP\Tomcat 4.1\webapps\CVP\audio\WeatherReport. Because our prompts are bilingual, we divide them into two subdirectories within this folder. French Canadian prompts go into a subdirectory called ca-fr, and US English prompts go into a subdirectory called us-en.

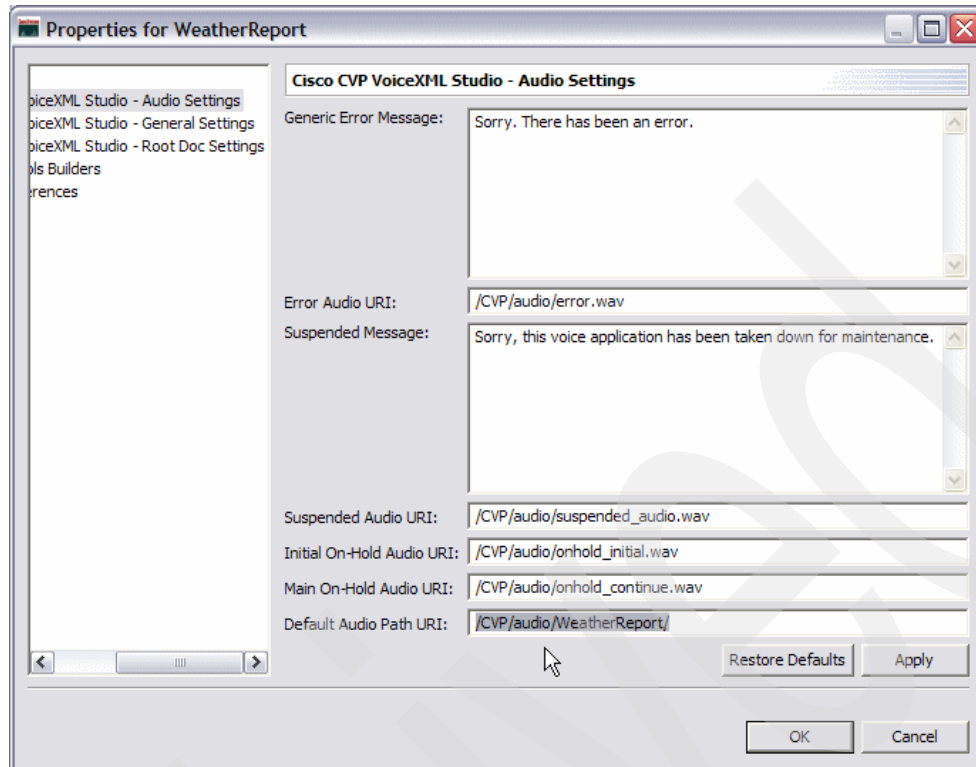


Figure 4-204 WeatherReport: Application Properties: Setting the Default Audio Path

Figure 4-205 on page 312 shows our application with the Select Language element highlighted. This is a 2-input Menu element. On the right pane, notice that the *Option DTMF* fields are set to **1** and **2** (the keys which would be pressed), but the Option Value fields are set to **en-us** and **ca-fr**. Once a caller enters either 1 or 2 here, the element value in the correct locale string format will be available to other nodes.



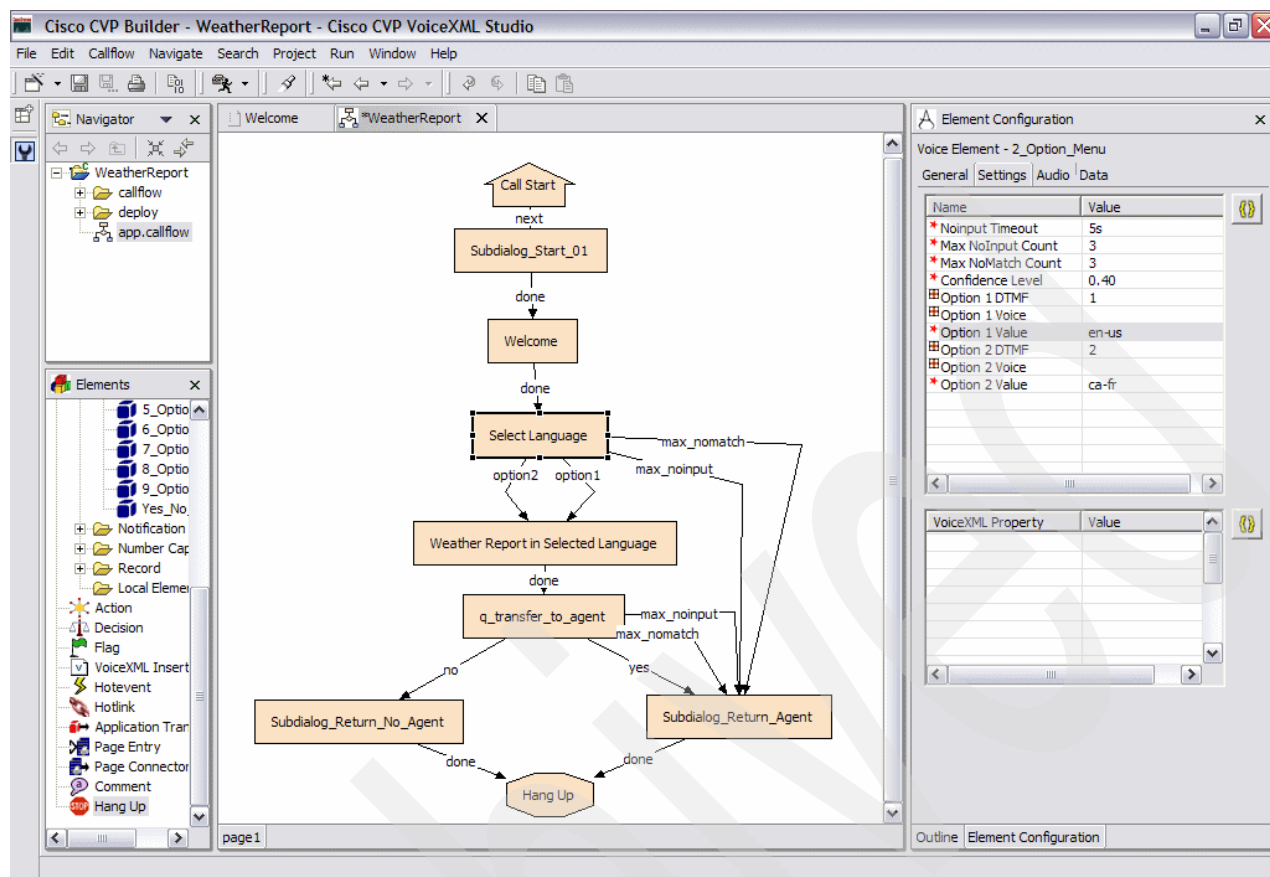


Figure 4-205 WeatherReport: Application Diagram showing Locale Selection

Figure 4-206 shows how the selected locale can now be embedded into the path name of any subsequent element. We see here that the weather report audio prompt will play from either the en-us/ or the ca-fr/ subdirectory.

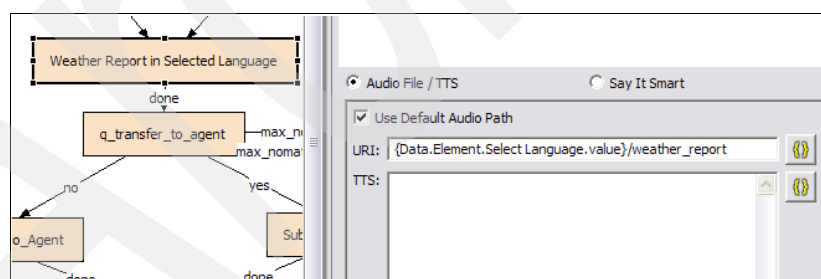


Figure 4-206 WeatherReport: Embedding Locale in the Audio Path

Data is returned to ICM by populating the various fields in the Subdialog\_return element. The value of Caller Input returns to the ICM routing script in the ECC variable user.microapp.caller\_input. The values in the External VoiceXML fields return to ICM in array elements 0 through 3 of the ECC variable user.microapp.FromExtVXML[].

Figure 4-207 on page 313 shows that we are returning the actual selected locale string in the caller\_input variable. Not shown here is the alternative Subdialog\_return node, in which the caller chooses not to be transferred to an agent. There the Caller Input field is set simply to 0.

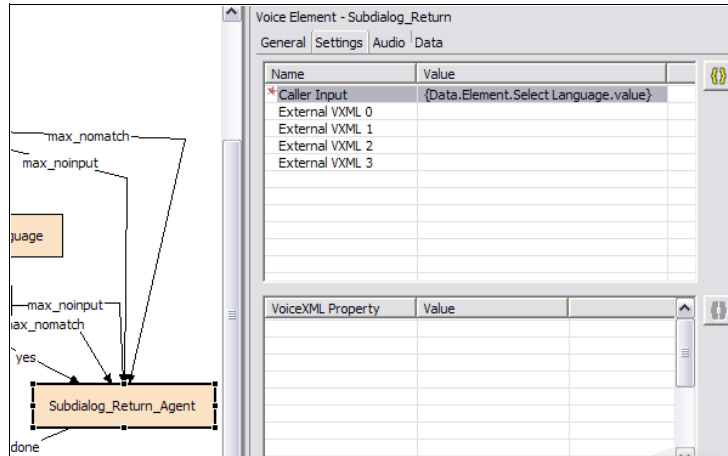


Figure 4-207 WeatherReport: Returning Locale to ICM

### 4.6.3 Configuring WeatherReport to run in standalone mode

Our application can now be tested in Standalone mode. The application will function correctly, but the returned data will not cause a transfer to an agent phone.

As before, the Application Service section (see Example 4-24) describes how the gateway should invoke the WeatherReport application on the VoiceXML Server. Note that both the server running Tomcat is identified as both the backup and the primary server VoiceXML Server. That is because we do not intend to deploy this application to the WebSphere Application Server based server.

*Example 4-24 Configuration added to the Cisco AS5400HPX Universal Gateway*

```
application
service WeatherReport flash:CVPSelfService.tcl
  paramspace english language en
  paramspace english index 0
  paramspace english location flash:
  param CVPBackupVXMLServer 9.42.171.24
  param CVPSelfService-app WeatherReport
  param CVPSelfService-port 8080
  paramspace english prefix en
  param CVPPrimaryVXMLServer 9.42.171.24
!
dial-peer voice 2002 voip
  translation-profile incoming block
  service weatherreport
  incoming called-number 2002
  dtmf-relay rtp-nte h245-signal h245-alphanumeric
  codec g711ulaw
  no vad
!
dial-peer voice 20021 pots
  service weatherreport
  incoming called-number 7772002
  direct-inward-dial
!
```

Dial-peers 2002 and 20021 also follow the pattern used before; 2002 enables us to call the application in standalone mode by dialing **2002** on an H323 SoftPhone, and 20021 allows us to call it by dialing **7772002** from the Gordon Kapes TDM simulator.

Again, remember to load the application by typing **call application voice load WeatherReport** on this IOS command line.

#### 4.6.4 WeatherReport ICM routing script

To set up ICM to handle this application, we first need to declare a Call Type and a Dialed Number. When a call is announced to ICM, these two items together select a particular routing script to execute.

Figure 4-208 on page 314 shows the ICM Configuration Manager window. We use both the Call Type list and the Dialed Number list tools.

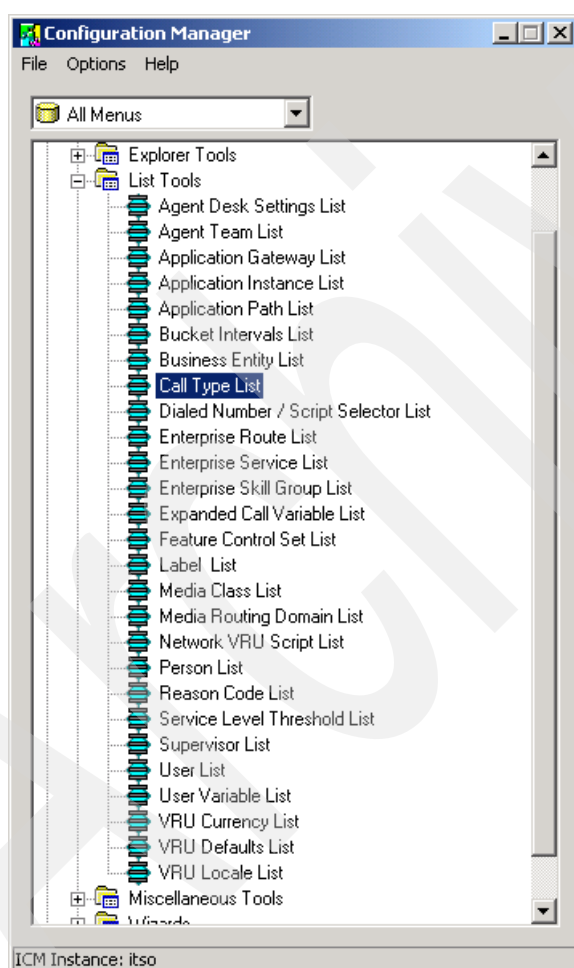


Figure 4-208 Using the ICM Configuration Manager to start the Dialed Number and Call Type list tools

1. Beginning with the Dialed Number list tool, create a new dialed number with the following settings, as shown in Figure 4-209 on page 315:
  - Routing client: **CVP**
  - Media routing domain: **Cisco\_Voice**

- Dialed number string: **7774001** (this will be the number we dial on the TDM phone)
- Customer: **ITSO**

Click **Save**.

Figure 4-209 Creating the Dialed Number in ICM

2. To create the Call Type, open the Call Type list tool. As shown in Figure 4-210, enter WeatherReport for the Name, and allow the remaining fields to default.

Click **Save**.

Figure 4-210 Creating a Call Type

3. Now we move to the ICM routing script. Open the ICM Script Editor by double-clicking the Script Editor icon in the ICM Admin Workstation program group, as shown in Figure 4-211 on page 316.

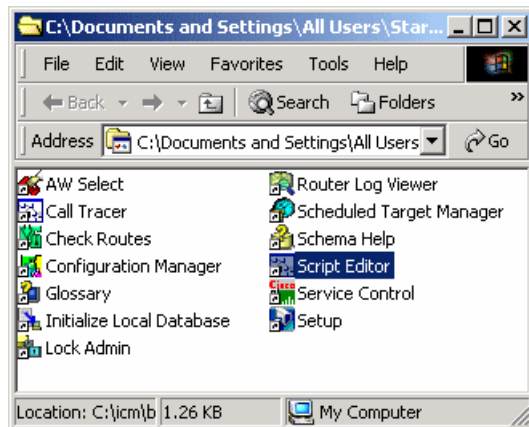


Figure 4-211 Starting the ICM Script Editor

4. Create a new routing script by clicking **File** → **New**, as shown in Figure 4-212. You are given the opportunity to select Routing script or Administrative script. Choose **Routing script**.

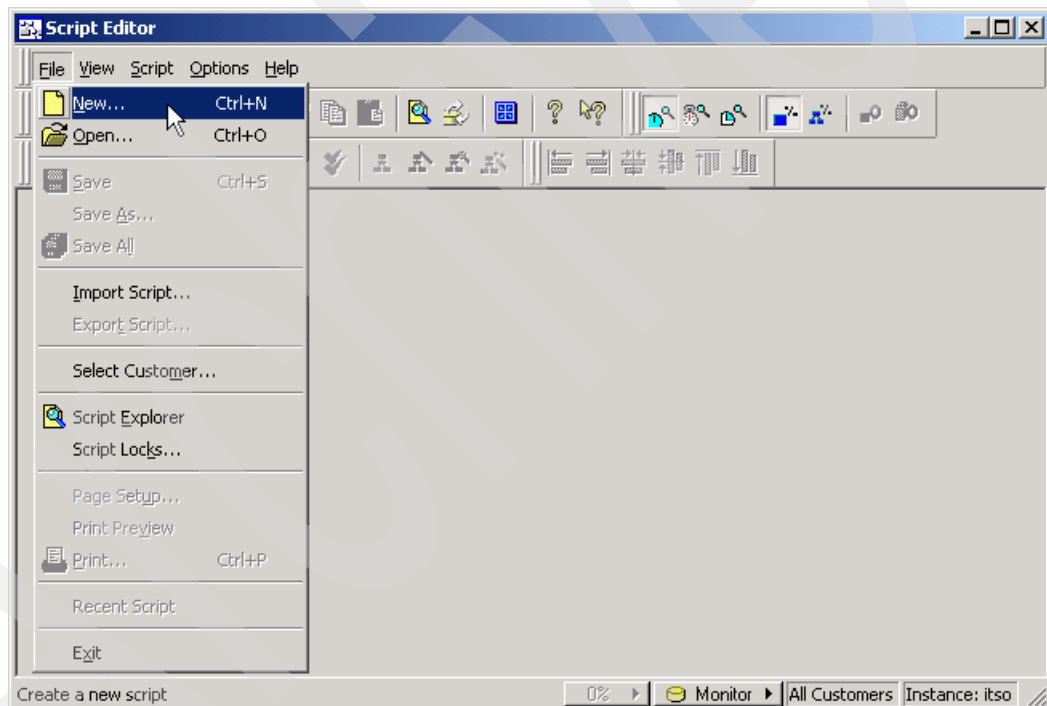


Figure 4-212 Creating a New ICM Routing Script

Figure 4-213 on page 317 shows the routing script that we created for this example. We do not describe the process of using the ICM Script Editor here, but we touch on several relevant script node configurations.

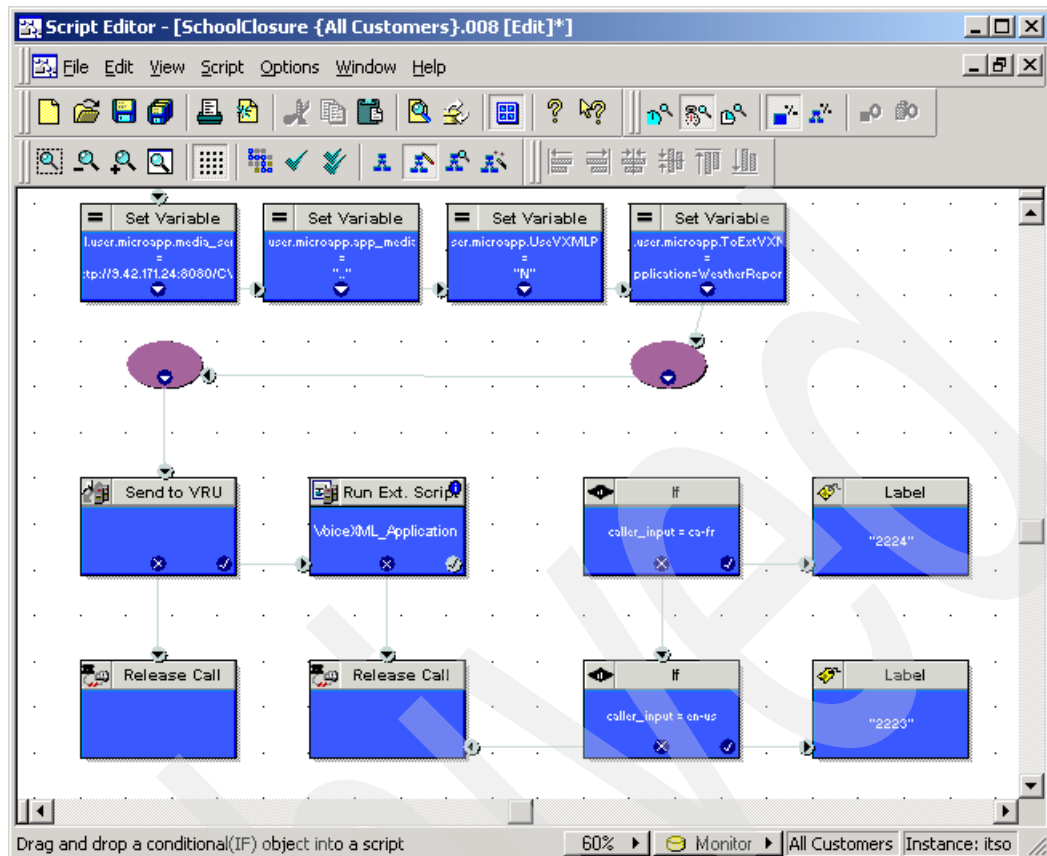


Figure 4-213 The ICM Routing Script

- Several ECC variables must be set to specific values at the start of any ICM routing script which will invoke CVP applications. These are listed in Table 4-18.

Table 4-18 ECC variables

ECC variable	Value
user.microapp.media_server	First part of the URL required to reach the CVP VoiceXML Server. We used <b>"http://9.42.171.24:8080/CVP"</b> , as shown in Figure 4-214.
user.microapp.app_media_lib	Must be <b>"."</b>
user.microapp.UseVXMLParams N	Must be <b>"N"</b>
user.microapp.ToExtVXML[0] application=WeatherReport	Points to the actual VoiceXML Server application to be invoked. As shown in Figure 4-215, we used <b>"application=WeatherReport"</b> .

- If your application interleaves CVP VoiceXML Server application calls with CVP Microapplication nodes, then you must continually modify the contents of user.microapp.media\_server and user.microapp.app\_media\_lib. The values used here are specifically for use when calling CVP VoiceXML Server applications.
- Also, you can use the user.microapp.ToExtVXML[] array to pass additional values to your CVP VoiceXML Server application. Values can be strung together in a single variable, separated by semicolons, as in:

"application=WeatherReport; Location=California"

Alternatively, you can split them into successive ECC array variable elements. See Figure 4-214 and Figure 4-215.

The 'Set Properties' dialog box has three tabs: 'Set Variable', 'Comment', and 'Connection Labels'. The 'Set Variable' tab is active. It contains the following fields:

- Object type:** A dropdown menu with 'Call' selected.
- Object:** A dropdown menu with '(No selection)' selected.
- Variable:** A dropdown menu with 'user.microapp.media\_server' selected.
- Array index:** An empty text input field.
- Value:** A text input field containing the URL 'http://9.42.171.24:8080/CVP'.

There are two 'Formula Editor...' buttons, one next to the 'Array index' field and one next to the 'Value' field. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

Figure 4-214 Pointing to the VoiceXML Server

The 'Set Properties' dialog box has three tabs: 'Set Variable', 'Comment', and 'Connection Labels'. The 'Set Variable' tab is active. It contains the following fields:

- Object type:** A dropdown menu with 'Call' selected.
- Object:** A dropdown menu with '(No selection)' selected.
- Variable:** A dropdown menu with 'user.microapp.ToExtVXML[]' selected.
- Array index:** A text input field containing the value '0'.
- Value:** A text input field containing the value 'application=WeatherReport'.

There are two 'Formula Editor...' buttons, one next to the 'Array index' field and one next to the 'Value' field. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

Figure 4-215 Selecting the VoiceXML Server Application

8. In the RunExtScript node, we reference the special CVP Microapplication which invokes the CVP VoiceXML Server application. Select the script named **VoiceXML\_Application**, which we created in Chapter 3, “Advanced deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3” on page 47. This is shown in Figure 4-216 on page 319. All routing scripts which invoke CVP VoiceXML Server applications can reference the same VRU script shown here.



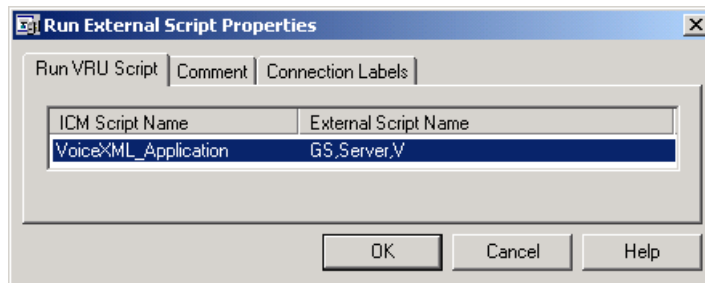


Figure 4-216 Selecting the VoiceXML Microapplication

When the WeatherReport application returns, it will contain a value in the `user.microapp.caller_input` ECC variable. The value will be either a valid locale (**en-us** or **ca-fr**), or the number **0**. If it is a valid locale, we must transfer the call to the appropriate travel agent. Otherwise, we can simply hang up. Figure 4-217 shows an example of how the necessary IF nodes would be configured.

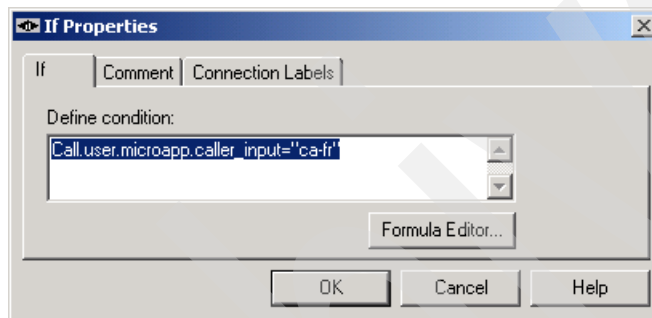


Figure 4-217 Configuring an IF node to branch based on locale

9. Finally, we save the script under the name `WeatherReport`, as shown in Figure 4-218.

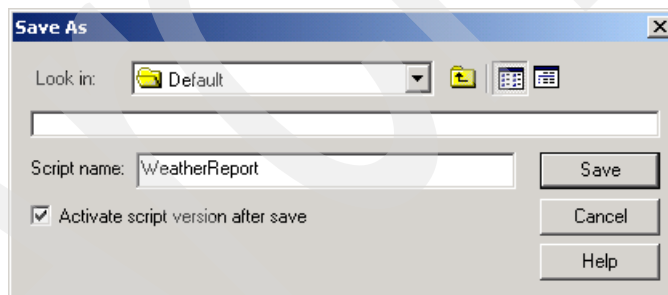


Figure 4-218 Saving the ICM Routing Script

Now we need to create the mapping between the dialed number and call type that we created for this example, and the routing script that we just saved. First we map the dialed number to the call type, then we map the call type to the script. The latter mapping also allows us to specify a schedule (days, hours, etc.) for which this mapping is valid; for our purposes we will accept the default of all day, every day.

Figure 4-219 shows how we start the ICM Call Type Manager. Note that it is a feature of the ICM Script Editor, not of the ICM Admin Workstation or the ICM Configuration Manager tools.

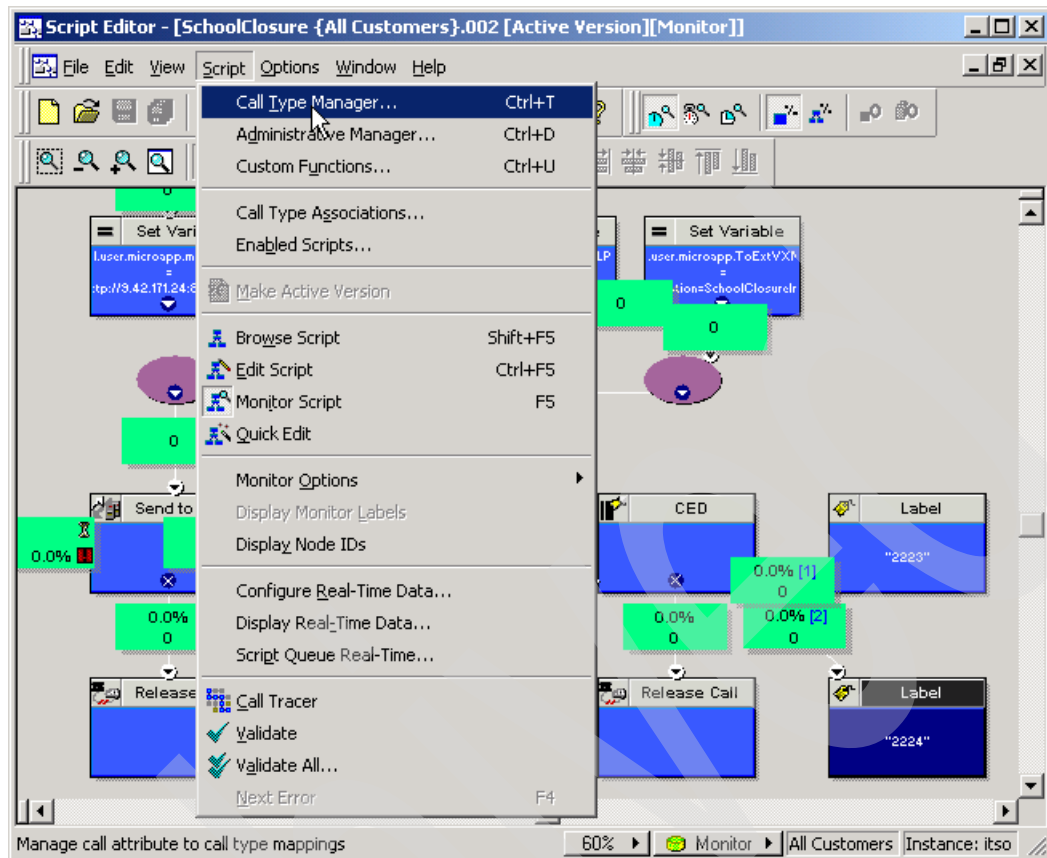


Figure 4-219 Starting the ICM Call Type Manager

1. On the **Call Directory** tab, select the Dialed Number we created earlier from the menu list, and click **Add** as shown in Figure 4-220 on page 321.

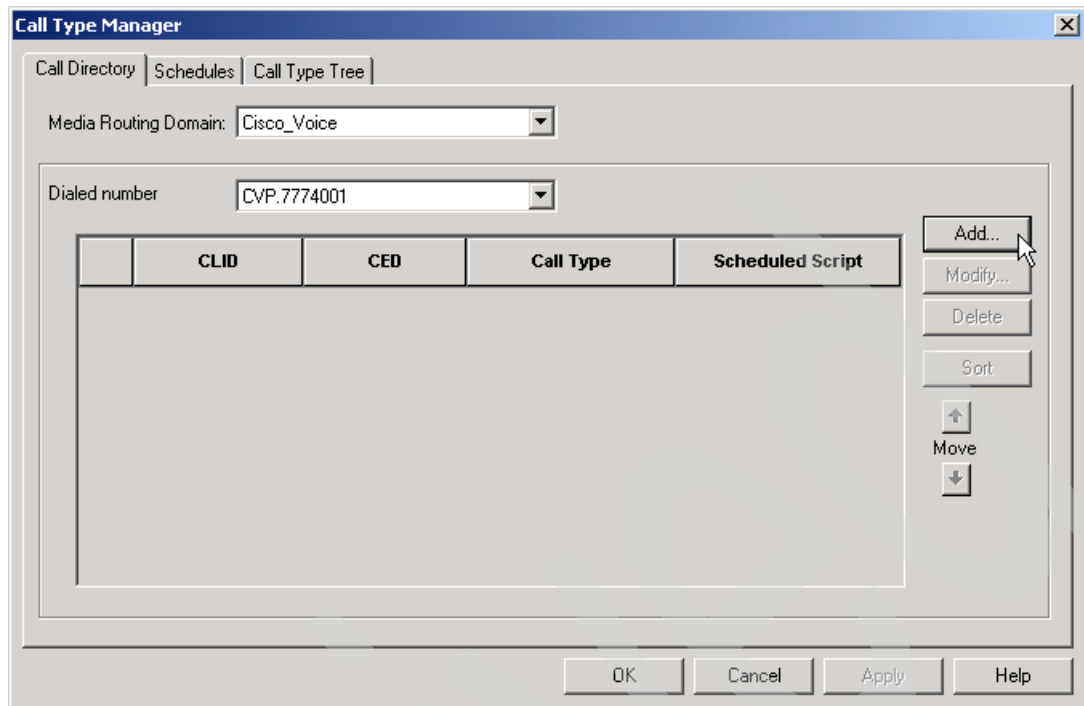


Figure 4-220 Adding a Dialed Number in the Call Type Manager

2. The Add Dialed Number Entry dialog box opens, as shown in Figure 4-221. The only thing we need to change here is the Call Type. Select **WeatherReport** from the menu list, and click **OK**.

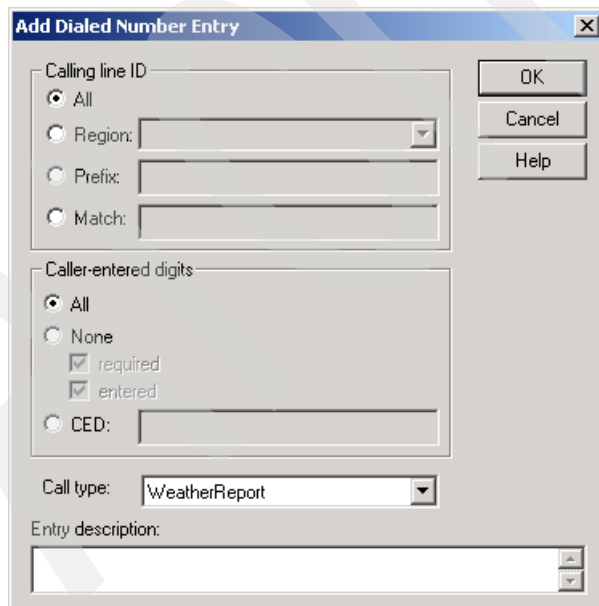


Figure 4-221 Linking a Dialed Number to a CallType

3. Back in the Call Type Manager dialog box, click the **Schedules** tab. Select **WeatherReport** from the Call Type menu list, and click **Add**. The Call Type Schedule dialog box opens, as shown in Figure 4-222. Map the WeatherReport Call Type to the

WeatherReport routing script by clicking on **WeatherReport** and pressing **OK**. We allow the more sophisticated scheduling settings to default to all day every day.

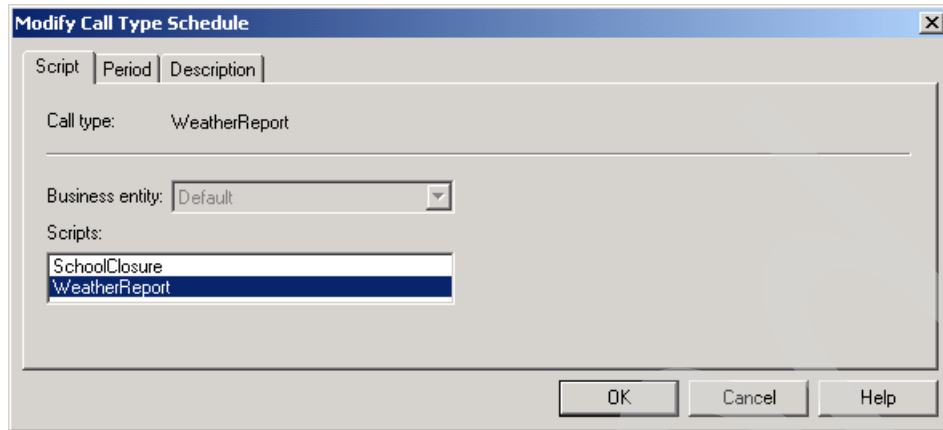


Figure 4-222 Call Type Schedule dialog box

At this point the Call Type Manager dialog box should look similar to Figure 4-223.

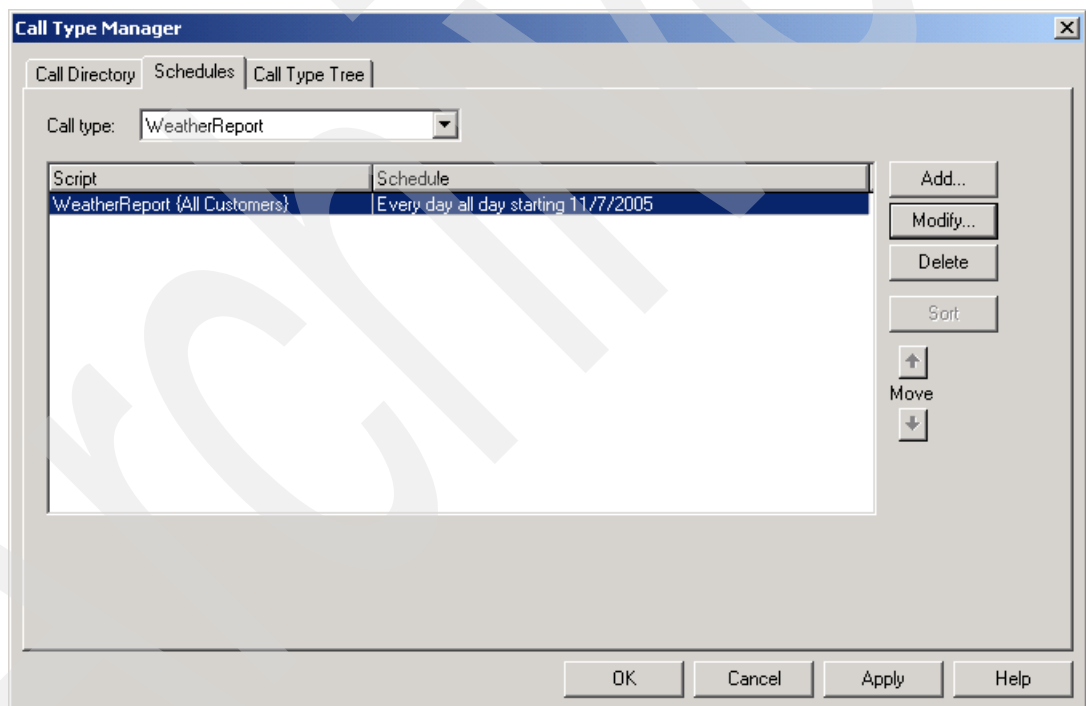


Figure 4-223 Scheduling a CallType to Run

This completes the ICM configuration necessary to invoke the ICM routing script, to have the ICM routing script invoke the CVP VoiceXML Server application, and to have the ICM routing script receive and act on the returned value.

Now we move to the Gateway and Gatekeeper configuration.

## 4.6.5 Configuring the Gateway and Gatekeeper

As it turns out, no additional Gateway or Gatekeeper configuration is necessary in order to support an additional ICM application. The generic configuration that we performed in Chapter 3, “Advanced deployment solution for Cisco CVP V3.1 and WebSphere Voice Server V5.1.3” on page 47 is good for all ICM Integrated applications.

However, we must configure the Gatekeeper to support *outgoing* calls to our two travel agents. We will start the H323 Softphone on two PCs. Each softphone must be configured to register its phone number with the Gatekeeper, and the Gatekeeper must be configured (see Example 4-25) to return the correct PC’s IP address when CVP tries to call those numbers.

*Example 4-25 Excerpt from the resulting Gatekeeper configuration*

```
gatekeeper
zone local Gatekeeper-3660 itso.ral.ibm.com 9.42.171.186
zone prefix Gatekeeper-3660 2223* gw-priority 10 9.42.171.106
zone prefix Gatekeeper-3660 2224* gw-priority 10 9.42.171.120
zone prefix Gatekeeper-3660 777* gw-priority 10 9.42.171.150
zone prefix Gatekeeper-3660 8887878* gw-priority 10 9.42.171.128
gw-type-prefix 1#* default-technology
no shutdown
```

Phone 2223 is our English agent, and 2224 is our French Canadian agent.

After both phones, the gateway, and the CVP Call Control Server are all registered, the gatekeeper shows the status, as seen in Example 4-26.

*Example 4-26 Gatekeeper status*

```
Gatekeeper-3660#show gatekeeper endpoints
GATEKEEPER ENDPOINT REGISTRATION
=====
```

CallSignalAddr	Port	RASignalAddr	Port	Zone Name	Type	Flags
9.42.171.106	1720	9.42.171.106	3842	Gatekeeper-3660	TERM	
E164-ID: 2223						
9.42.171.120	1720	9.42.171.120	2438	Gatekeeper-3660	TERM	
E164-ID: 2224						
9.42.171.128	1720	9.42.171.128	52484	Gatekeeper-3660	VOIP-GW	
H323-ID: 9.42.171.128						
Voice Capacity Max.= Avail.= Current.= 0						
9.42.171.150	1720	9.42.171.150	1719	Gatekeeper-3660	VOIP-GW	
H323-ID: 9.42.171.150						
Voice Capacity Max.= Avail.= Current.= 0						
Total number of active registrations = 4						

## 4.6.6 Executing the WeatherReport application

We are now ready to call the application. Note that because this is an ICM Integrated application, we can only run it using a TDM phone. H323 Softphones cannot be used.

From a TDM phone, dial **7774001**. This invokes the ICM routing script, which in turn invokes the WeatherReport application on the CVP VoiceXML Server. When the application asks you to select a language, press either 1 and 2 alternatively. The language of subsequent voice prompts switches from English to French.

When the application asks whether you wish to transfer to a travel agent, press 1 (yes). If your chosen language is English, you should see phone 2223 ring. If your chosen language is French, you should see phone 2224 ring. If you press 2 (no), then the application will disconnect automatically.

Clearly, this example application does not fully demonstrate the power of ICM Integration. In a real contact center, the ICM would likely choose an agent from among thousands of available agents who spoke the selected language. Those agents might be located at different call centers around the world, and connected using IP phones, TDM phones, ACDs, home phones, and so forth. If no appropriate agents were currently available, then different skill groups could be tried. Failing that, the call could be queued until the right agent was available. Additionally, more than likely, the self-service application would return considerably more data than just the language selection, and ICM would make that data immediately available to the agent who receives the phone call with a screen pop-up. All this, however, is well beyond the scope of this Redpaper.

## 4.7 IBM WebSphere Voice Toolkit V6.0.1

The WebSphere Voice Toolkit V6.0.1 helps you create complex voice applications quickly and easily. It is powered by Eclipse technology, making it easier to develop VoiceXML applications without having to know the internals of voice technology.

Within the product are tools to build, debug, and deploy voice applications across the IBM WebSphere Voice family:

- ▶ WebSphere Voice Application Access
- ▶ WebSphere Voice Server,
- ▶ WebSphere Voice Response.

Some of the features of the WebSphere Voice Toolkit features include:

- ▶ A full-featured voice development environment including graphical call flow building
- ▶ VoiceXML development and debugging
- ▶ Grammar development and debugging
- ▶ Pronunciation builder
- ▶ Call Control extensible Markup Language (CCXML) development

Communication Flow Builder, is an easy to use graphical tool for visual composition of a voice application, allows for the design and prototype of a voice application by someone who is not familiar with VoiceXML programming. The Communication Flow Builder is an enhanced version of the Call Flow Builder.

The IBM WebSphere Voice Toolkit includes a XML-based Lexicon editor that provides the ability to create a file of customized pronunciations for TTS or Recognition, a migration utility for converting pronunciation files produced with previous versions of the Voice Toolkit to the new Lexicon format, and an MRCP-based Grammar Test Tool to load and test grammars and lexicon files with WebSphere Voice Server V5.1.X.

New in the WebSphere Voice Toolkit V6.0:

- ▶ Migration of the Voice Toolkit to Rational Software Development Platform V6.0.
- ▶ Graphical Grammar Builder for visual composition of a grammar file for speech recognition.
- ▶ Communication Flow Builder which is an enhanced version of Call Flow Builder that includes support of Reusable Dialog Components from the Apache Open Source group.

- ▶ Prompt Manager for organizing the Audio Files in a voice application.
- ▶ Voice Trace Analyzer which analyzes log files from the WebSphere Voice Server V5.1.X
- ▶ VoiceXML V2.1 support (based on W3C standards)
  - Future enhancements will include updated conversion tools as well as enhanced development, simulation, and debug environments
- ▶ Enhanced VoiceXML Simulation and Debug capabilities using a SIP-based phone and a MRCP Server
- ▶ Enhanced Grammar and Audio Conversion tools

Following the IBM WebSphere Voice Toolkit installation documentation.

## 4.8 IBM WebSphere Voice Toolkit V6.0.1 enhanced features

For this Redpaper, the VoiceXML application was developed using Cisco's CVP V3.1 studio product. However, we used some of the enhanced features of the toolkit to compliment the voice application. Specifically, we used:

- ▶ Lexicon pronunciation editor
- ▶ Grammar creation
- ▶ Reusable Dialog Components or RDC

### 4.8.1 Lexicon pronunciation editor

The lexicon editor creates pronunciation information for both the IBM speech recognition and text-to-speech engines. It does this by using a standard pronunciation alphabet to aid the developer in creating a pronunciation for a word or phrase. These pronunciations are then grouped together in a *lexicon*. The voice application then references this by using markup languages.

#### Creating a Lexicon file

To create a Lexicon file follow these steps.

1. If you have not done so, create a voice project. From the file menu, click **New** → **Voice Project**.
2. Create a folder to place the Lexicon file.
  - a. Select **File** → **New** → **Other** → **Simple** → **Folder** and click **Next**.
  - b. Choose the WebContent folder, type a name and click **Finish**.
3. From File menu, select **New** → **Lexicon file**.
4. Select whether to create the Lexicon file from a DTD file or from scratch. We choose the **DTD** file. Click **Next**.
5. Type the new file name. The default extension for a Lexicon file is .lxml.
6. There are two radio buttons. Select the type of file:
  - **Recognition** is used to create pronunciations for the speech recognition engine using International Phonetic Alphabet (IPA) phonologies. This is used to create grammars for words the user is expected to say when interacting with the application. This is the default.



- **Synthesizer or Text-To-Speech** is used to create pronunciations for the Text-To-Speech engine using Symbolic Phonetic Representations (SPR) phonologies. This is used to create grammars for words the synthesized voice speaks to users.
- 7. Optional: The Advanced option provides the ability to link to a file in the file system.
- 8. Click **Finish**. The Lexicon editor launches your file with the basic <lexicon> tag.

### Voice Toolkit configuration to a working WebSphere Voice Server V5.1.3

To make the ASR and TTS function within the Toolkit, a valid IP address of a working voice server is required. This is configured in **Preferences** → **Speech Engines** → **WebSphere Voice Server V5.1 (MRCP)**. Figure 4-224 displays the working configuration used in the ITSO Lab.

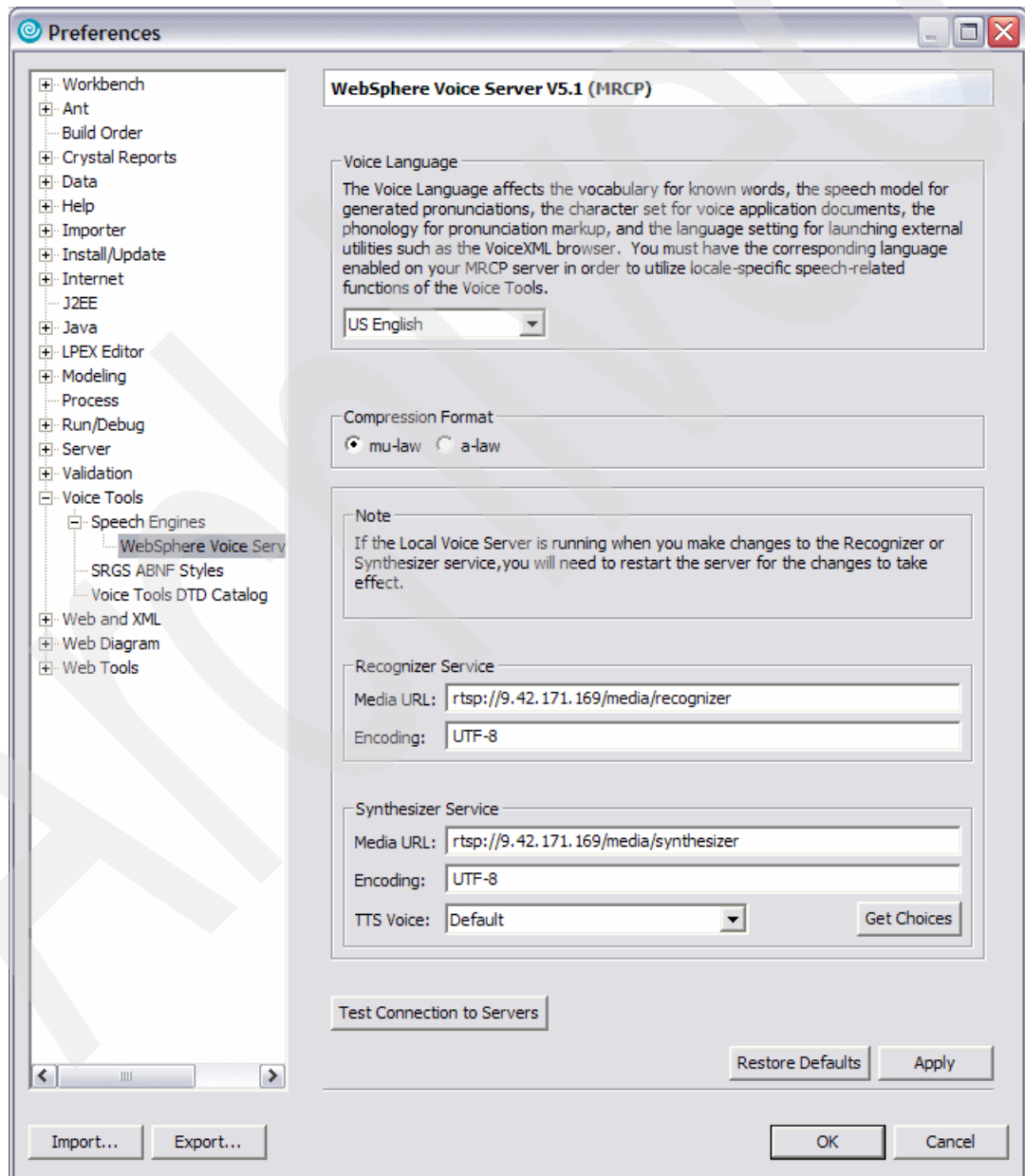


Figure 4-224 Speech Engine configuration within the toolkit

## Creating an ASR Lexicon pronunciation

In our school application, the name Durham was incorrectly pronounced. The Lexicon editor was used to create a created spoken version. Figure 4-225 shows where the school name was typed.

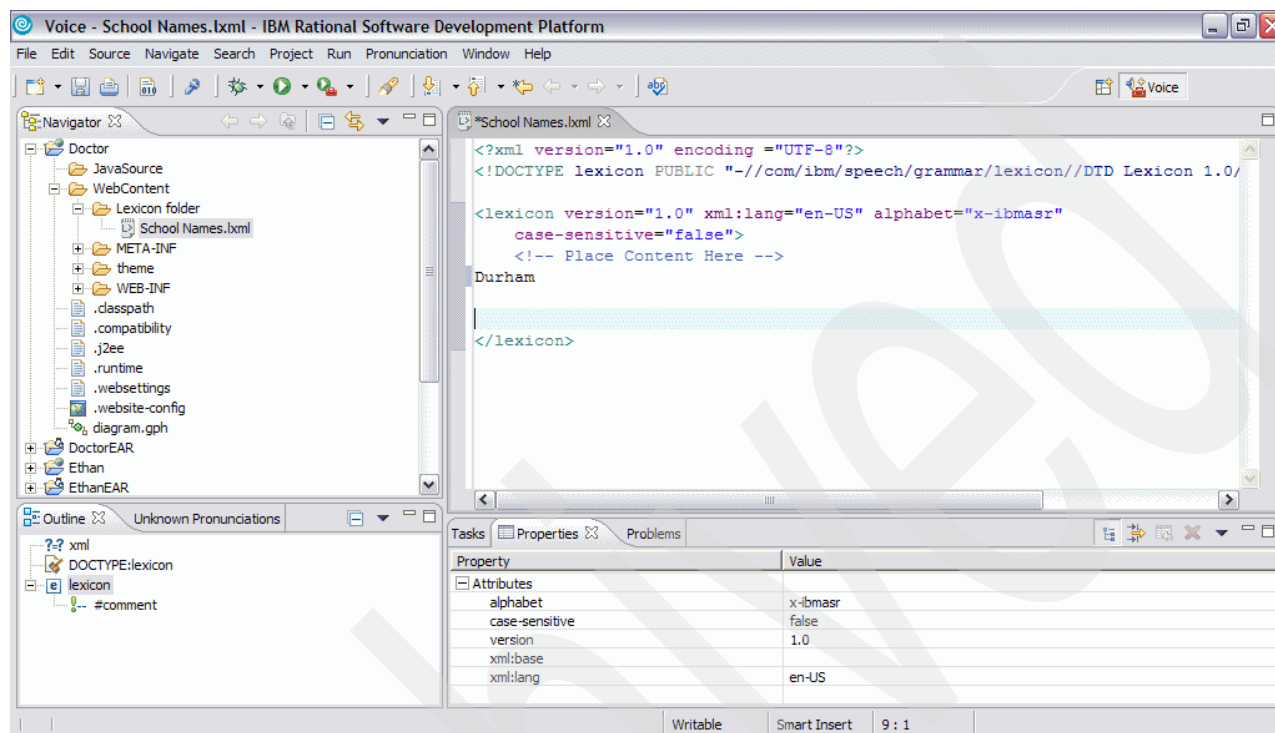


Figure 4-225 Lexicon editor

1. Highlight and right-mouse click **select Compose Pronunciation**. Figure 4-226 on page 327 has the Pronunciation Builder. Also displayed is the default Recognition Pronunciation, which in this case is wrong.

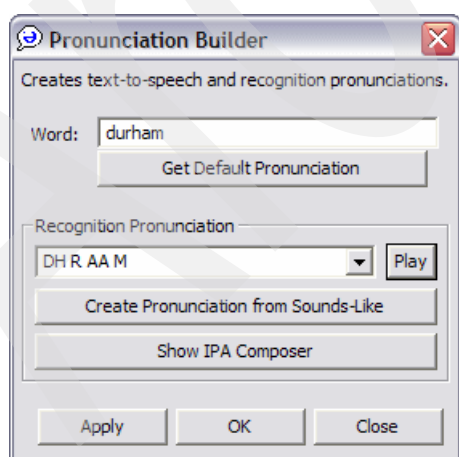


Figure 4-226 Lexicon Pronunciation Builder with default recognition pronunciation

2. Click **Show IPA Composer**. Here you select the letters to construction a correctly pronounced word. In Figure 4-227 the final version of the word Durham can be seen. At any stage, you can play the word back to see how accurate it is.

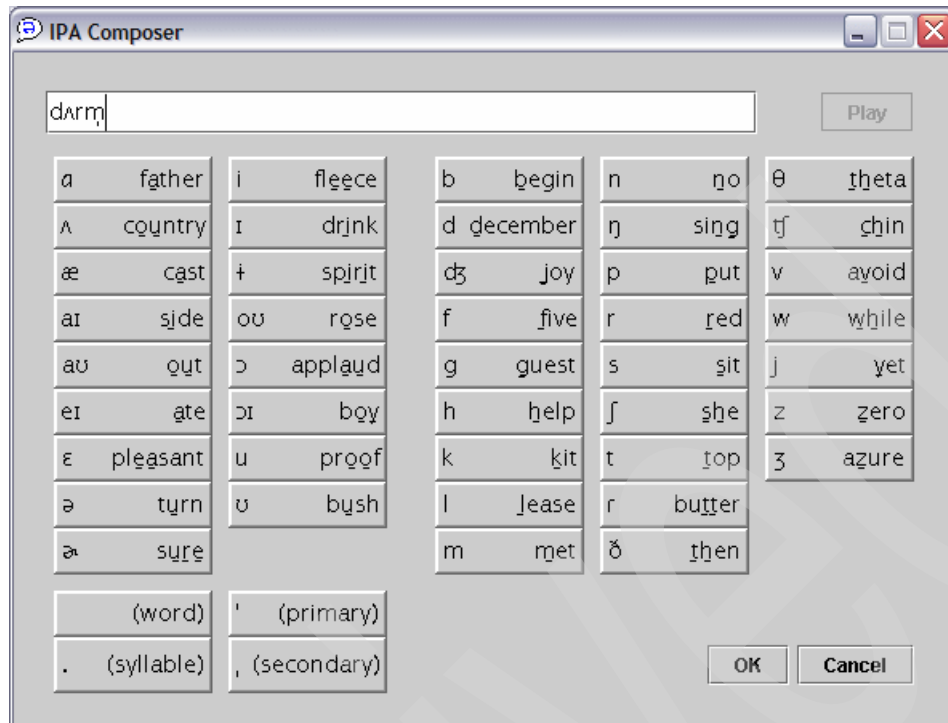


Figure 4-227 IPA composer

- When you are happy with the way the pronunciation sounds, click **OK**, and then click **OK** again in the Pronunciation Builder. There is a new entry in the lexicon file for this word. See Figure 4-228. When you have finished, save the file. You can then export to a file ready to be imported into the CVP V3.1 studio project.

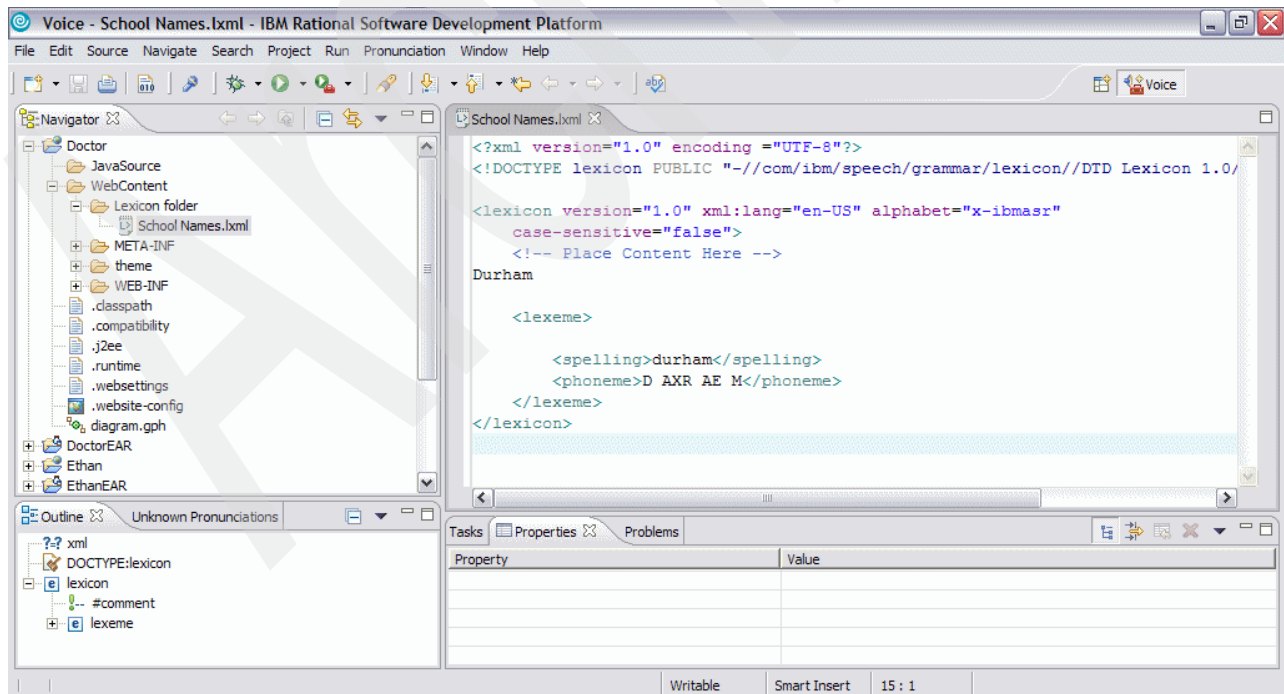


Figure 4-228 Lexicon data from the Pronunciation Builder

Remember, it is possible to have multiple pronunciations of the one word. This can be necessary when there are several ways to say it, perhaps due to different accents. The final code can be seen in Example 4-27.

*Example 4-27 Lexicon file*

---

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE lexicon PUBLIC "-//com/ibm/speech/grammar/lexicon//DTD Lexicon 1.0//EN"
"ibmlexiconml.dtd">

<lexicon version="1.0" xml:lang="en-US" alphabet="x-ibmasr"
  case-sensitive="false">
  <!-- Place Content Here -->
  Durham

    <lexeme>

      <spelling>durham</spelling>
      <phoneme>D AXR AE M</phoneme>
    </lexeme>
  </lexicon>
```

---

## 4.8.2 Creating an external grammar file

In some instances a complex grammar or a grammar that will be reused in the application maybe needed. One approach is to write an external SRGS XML grammar and reference it in the application. To create this external SRGS grammar file follow these steps:

1. In the toolkit select **File** → **New** → **SRGS XML Grammar File**.
2. Select **Create GRXML file from a DTD**. Select **Next**.

**Note:** DTD stands for *Document Type Definition*. It provides a set of declarations that conform to a particular markup syntax and describe a type of XML document.

3. Enter a name for the grammar file, ensuring the file extension is .grxml. Click **Finish** to add the grammar file to the project directory.

## 4.8.3 Reusable Dialog Components

This next section describes using Reusable Dialog Components (RDCs) in a Cisco CVP Studio Application. For additional information about RDC, please see *Speech User Interface Guide, REDP-4106*.

### Using an RDC in a CVP Studio Application

Reusable Dialog Components (RDCs) are a framework that leverages server-side code generation to accelerate the development of voice applications. RDCs are similar in functionality to CVP Studio callflow elements in that they encapsulate a particular interaction (for example, asking the caller to enter a telephone number). It is possible to include RDCs in your CVP Studio applications, as in the SchoolClosureInfo application. This provides CVP Studio developers with access to the wide range of existing RDCs for inclusion in their callflows, as well as new RDCs as they become available in the future.

There are two methods for including RDCs within a CVP Studio application. First, a VoiceXML Insert element can be used, which executes the RDC as a subdialog. This approach requires you to create an external VoiceXML file that does the work of calling the

RDC. The second option is to generate the VoiceXML dynamically, through a custom voice element that acts as a wrapper for the RDC. The functionality of this approach is very similar to using a VoiceXML Insert element, except that the VoiceXML is generated at runtime using the Voice Foundation Classes (VFCs). This option has the added benefit of the custom element appearing in the CVP Studio Elements pane, for easy drag-and-drop access.

Both of these options have an additional requirement, due to the nature of the RDCs. Because the RDCs must be called with special JSP™ tags, a JSP page must be created that performs the RDC call. The flat (with VoiceXML Insert) or dynamic (with custom voice element) VoiceXML code calls this JSP page as a subdialog. After the subdialog is complete, the return value of the RDC is stored in either element or session data. The interaction is shown in Figure 4-229 on page 330. This is a conceptual diagram, the dotted line indicates a request that actually originates from the voice browser.

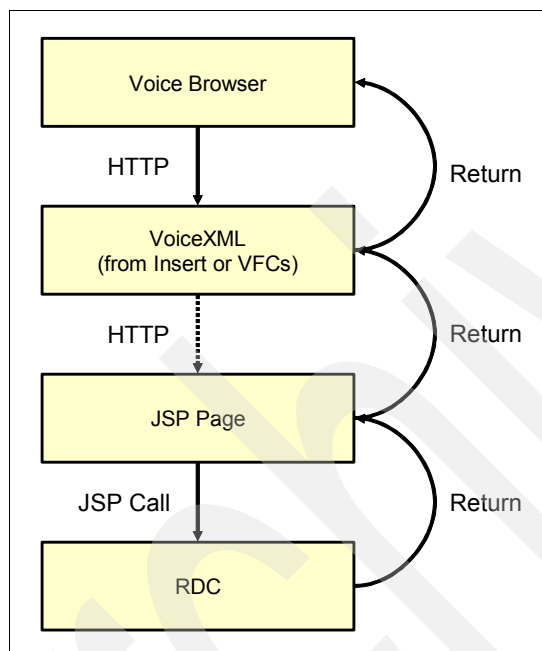


Figure 4-229

Accessing an RDC with a VoiceXML Insert element is simple to implement, and requires no custom Java code. VoiceXML Insert elements allow the voice application developer to specify the URI of an external VoiceXML file to access at runtime. A flat VoiceXML page is then created that calls an RDC through the previously discussed JSP page. However, since the VoiceXML is hard-coded, cross-browser compatibility cannot be guaranteed. Normally, all VoiceXML is produced by CVP Server, which filters it through the gateway adapter so that it is properly formatted for the selected voice browser. However, VoiceXML Insert elements bypass this functionality, since the VoiceXML is already pre-written by the developer. If the application is meant to run in only one environment, this might not be an issue.

In this solution, both the VoiceXML and JSP file must be hosted on an application server. Additionally, the application they are hosted under must include the RDC taglib, so that the JSP file will be able to access the RDCs at runtime. Information about installing the RDC taglib in a Web or voice application can be found here (see the Configuration section):

<http://jakarta.apache.org/taglibs/doc/rdc-doc/rdc-1.0/index.html>

If a given RDC will be used either in several different places throughout a single voice application's callflow or across multiple voice applications, it can be worthwhile to create a custom voice element that encapsulates it. This approach allows the element to be easily

dragged into callflows from the Elements pane of CVP Studio. The process involved in creating a wrapper element is somewhat more involved than using a VoiceXML Insert element, because it requires custom Java code.

This solution functions identically to the VoiceXML Insert element solution, but the VoiceXML is not hard-coded. It is generated at runtime by CVP Server using the VFCs. This helps ensure that your voice application will be compatible with any voice browser supported by CVP Server. However, there is one caveat. Because the RDCs generate their own VoiceXML, they are not certified to work on the same voice browsers as CVP Server. It is important to ensure that the voice browser being used supports the VoiceXML generated by the RDCs.

Wrapping the RDC in a custom voice element also includes the benefit of allowing the application designer to specify configuration values for the RDC from directly within CVP Studio. After the custom element is dragged into a callflow, it can be configured with settings that the RDC expects. For example, the usState RDC could be sent values for its locale, maxNoInput, maxNoMatch, confirm and other settings. In order to support these configurable settings, they should be included in the custom Java code by the voice element developer. Since it is the JSP page that performs the RDC call, these configuration options need to be passed from the dynamic VoiceXML produced by the custom voice element to the JSP page as subdialog parameters, and then passed as arguments to the RDC call.

For the SchoolClosureInfo application, the usState RDC is accessed via a VoiceXML Insert element. Because this RDC is accessed only once in the application, and did not need to be reused across multiple applications, it was simpler to implement in this fashion. Creating a custom voice element for this single-use element would have been excessive. Note, however, that if this element was to appear many times (either in this application or others), it might have been worth the time investment to create a reusable custom element.

Archived



## Testing and troubleshooting

This chapter provides a detailed guide on how to troubleshoot problems with WebSphere Voice Server V5.1.x and Cisco CVP VoiceXML Server V3.1. We will look at some of the tools provided and how to diagnose typical problems.

Troubleshooting or problem determination encompasses a wide range of tasks that might need to be performed at any phase in product usage. To do this information needs to be collected, analyzed and the findings investigated. To aid this process there are a variety of tools, services and features such as, but just including:

- ▶ Admin console
- ▶ Logs, traces, console events
- ▶ Collector tool.
- ▶ Test applications.
- ▶ Problem determination methodology

Because this solution is a multivendor platform, each component has its own troubleshooting tools and process. We look at some of these.

Covered in this chapter are installation verification testing procedures and post-installation problem determination.

## 5.1 Installation testing

The ITSO deployment solution was built in stages. The first stage was to build and configure the Cisco Customer Voice Portal CVP implementation. This involved

- ▶ Two CVP VoiceXML servers, each using a different application server
  - With Tomcat
  - With WebSphere Application Server
- ▶ Cisco 1751 router
- ▶ Database server, using MySQL

### 5.1.1 Cisco CVP phone call test

Before installing WebSphere Voice Server we tested the Cisco environment to ensure basic DTMF call handling worked. A simple VoiceXML application was used, which did not call and ASR or TTS functions from the Voice browser. It used caller interaction via DTMF functions. Figure 5-1 shows the simple call flow used. This was developed and deployed using Cisco CVP studio. The Cisco CVP studio project is available from the Redbook Web site.

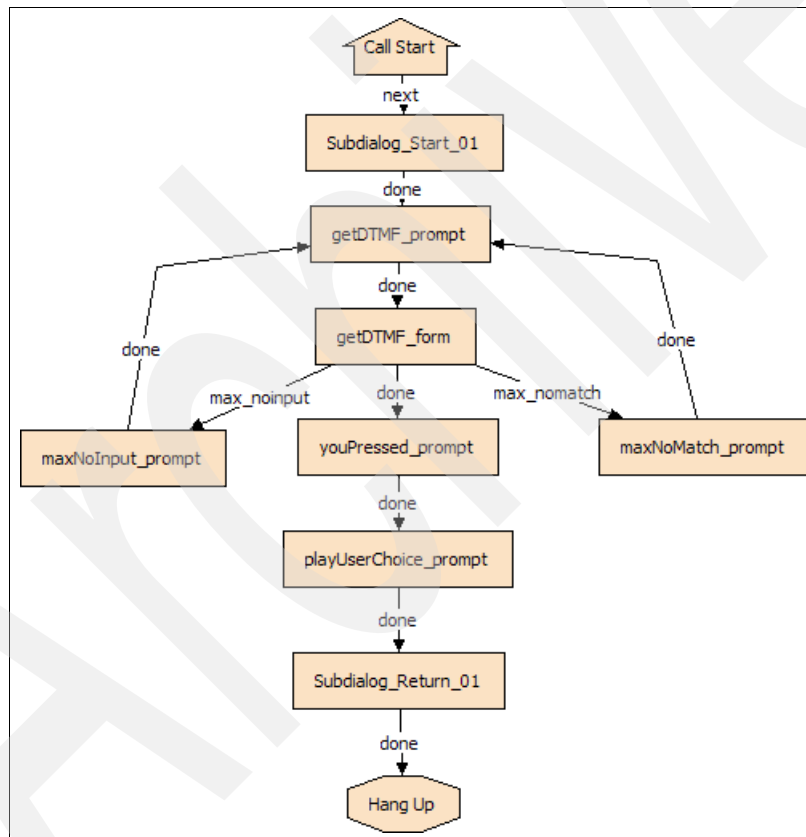


Figure 5-1 Simple DTMF call flow

The CVP VoiceXML server will create a dynamic VoiceXML file which is then sent back to the router. Example 5-1, is the VoiceXML code generated for the DTMFTest1 application.

#### Example 5-1 DTMFTest1 VoiceXML application

```
<?xml version="1.0" encoding="UTF-8" ?>
- <vxml version="2.0" application="/CVP/Server?audium_root=true&calling_into=DTMFTest1"
xml:lang="en-us">
```

```

- <form>
  <var name="testParam" />
- <block>
  <assign name="audium_vxmlLog" expr="''" />
  <submit next="/CVP/Server" method="post" namelist="testParam audium_vxmlLog" />
</block>
</form>
</vxml>

```

---

There are two logs that provide information about requests made to the Cisco CVP VoiceXML Server. The files names contain the date to help track any potential problems. They are located in:

```

C:\Cisco\CVP\Server\logs
C:\Cisco\CVP\Server\Applications\DTMFTest1\logs

```

Example 5-2 has the call\_log2005-10-28.txt from the server logs folder.

*Example 5-2 call\_log2005-10-28.txt*

---

```

9.42.171.24.1130516611375.10,9.42.171.24.1130516611375.10.DTMFTest1,0,1,DTMFTest1,10/28/200
5 12:23:31

```

---

Example 5-3 has matching activity\_log2005-10-28.txt from the application logs folder. This example is from the application folder called DTMFTest1.

*Example 5-3 activity\_log2005-10-28.txt*

---

```

9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:31,,start,newcall,
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:31,,start,ani,NA
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:31,,start,areacode,NA
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:31,,start,exchange,NA
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:31,,start,dnis,1001
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:31,,start,iidigits,NA
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:31,,start,uui,NA
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:31,Subdialog_Start_01,element,enter,
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:31,Subdialog_Start_01,element,exit,done
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:31,getDTMF_prompt,element,enter,
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:31,getDTMF_prompt,interaction,audio_group,initial_audio_group
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:31,getDTMF_prompt,element,exit,done
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:31,getDTMF_form,element,enter,
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:31,getDTMF_form,interaction,audio_group,initial_audio_group
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:39,getDTMF_form,interaction,noinput,1
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:39,getDTMF_form,interaction,audio_group,noinput_audio_group
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:46,getDTMF_form,data,value,2
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:46,getDTMF_form,data,result,2
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:46,getDTMF_form,data,confidence,1
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,getDTMF_form,data,value_confidence,1
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,getDTMF_form,data,nbestConfidence1,1
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,getDTMF_form,data,nbestUtterance1,2

```

---

```

9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,getDTMF_form,data,nbestInputmodel,dtmf
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,getDTMF_form,data,nbestInterpretation1,2
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:46,getDTMF_form,data,nbestLength,1
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:46,getDTMF_form,element,exit,done
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,decideWhichPromptClass,element,enter,
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,decideWhichPromptClass,element,exit,done
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:46,youPressed_prompt,element,enter,
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,youPressed_prompt,interaction,audio_group,initial_audio_group
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,youPressed_prompt,element,exit,done
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,playUserChoice_prompt,element,enter,
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,playUserChoice_prompt,interaction,audio_group,initial_audio_group
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,playUserChoice_prompt,element,exit,done
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005
12:23:46,Subdialog_Return_01,element,enter,
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:49,,end,how,app_session_complete
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:49,,end,result,normal
9.42.171.24.1130516611375.10.DTMFTest1,10/28/2005 12:23:49,,end,duration,17

```

---

## 5.1.2 Cisco CVP call test requesting WebSphere Voice Server ASR and TTS functions

At this point, it is assumed WebSphere Voice Server has been installed and configured. This includes the Cisco router configured to point to both a WebSphere Voice Server ASR server and a WebSphere Voice Server TTS server. In fact the Cisco settings for the ASR and TTS servers could be one of the following configurations:

- ▶ A single WebSphere Voice Server server running ASR and TTS
- ▶ Two WebSphere Voice Server Servers, one running ASR the other running TTS
- ▶ A WebSphere Load Balancer which in turn redirects MRCP requests to multiple WebSphere Voice Server servers in a balance fashion.

A simple VoiceXML application was used for testing. Figure 5-2 on page 337 shows the application call flow. In this case, the caller says a color invoking the ASR server. Then the response is analyzed and read back to the caller using the TTS server. The application was developed and deployed using Cisco CVP studio. The Cisco CVP studio project is available from the Redbook Web site.

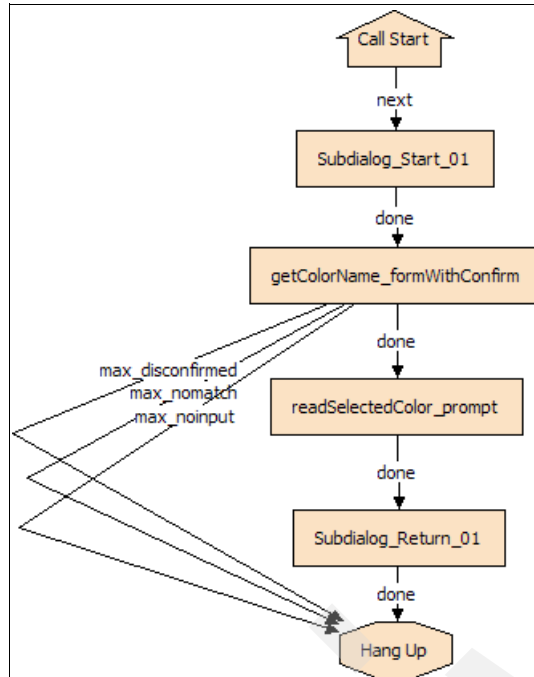


Figure 5-2 Simple ASR and TTS function call flow

The CVP VoiceXML server will create a dynamic VoiceXML file which is then sent back to the router. Example 5-4, is the VoiceXML code generated for the HelloWorld application.

**Example 5-4 HelloWorld VoiceXML application**

```

<?xml version="1.0" encoding="UTF-8" ?>
- <vxml version="2.0" application="/CVP/Server?audium_root=true&calling_into=DTMFTest1"
xml:lang="en-us">
- <form id="audium_start_form">
- <block>
  <assign name="audium_vxmlLog" expr="" />
  <assign name="audium_element_start_time_millisecs" expr="new Date().getTime()" />
  <goto next="#start" />
</block>
</form>
- <form id="start">
- <block>
- <prompt bargein="true">
  <audio src="/CVP/audio/pleasePress.wav" />
</prompt>
  <assign name="audium_vxmlLog" expr="audium_vxmlLog + '|||audio_group$$$' +
'initial_audio_group' + '^^^' +
application.getElapsedTime(audium_element_start_time_millisecs)" />
  <submit next="/CVP/Server" method="post" namelist="audium_vxmlLog" />
</block>
</form>
</vxml>
  
```

### 5.1.3 Testing Load Balancer within Cisco deployment

The complex deployment utilized WebSphere Edge Server as a load balancer. When an MRCP request from the Cisco router was made to the Cluster, 9.42.171.89, the Edge Server would balance the workload across the available WebSphere Voice Server machines.

It is important this is configured correctly. The WebSphere Voice Server REDBOOK xxxx, has a step by step instruction on how to set this up, and also to troubleshoot the deployment. This is a summary of what should be checked.

- ▶ All servers in the deployment WebSphere Voice Server, CVP, LB must have static IP addresses.
- ▶ Ensure that all host names for all the WebSphere Voice Server machines can be resolved to their IP address.
- ▶ Ensure each WebSphere Voice Server machine can ping:
  - Router
  - Each other
  - Edge server
- ▶ Do the same step, but this time from the Router and the Edge Server.

**Attention:** If your network is not using a DNS, ensure all the host names are resolved on each machine running WebSphere Voice Server and the edge server. Refer red book on how to do this. If this is not done correctly the WebSphere Voice Server traffic will not be handled correctly.

- ▶ Ensure the loopback adapter is configured on all WebSphere Voice Server machines, regardless of operating system. This is a very important step.
- ▶ All servers within the Cisco CVP WebSphere Voice Server deployment must be on the same segment, and do not pass through any routers or bridges. Latency will be experienced otherwise.

During our residency labwork, one of our WebSphere Voice Server servers lacked a loopback adapter. However, it was still configured in the cluster. This caused a strange result. The caller heard the initial TTS played, but then experienced a disconnection.

The initial MRCP request was sent to the voice server, but then the call was dropped. However, the session remained active to the WebSphere Voice Server until it timed out. Then the session was terminated. A second call was placed and it too created a second orphaned active connection. We used the Edge Server Monitor, and in this case specifically the Active Connection option. This monitor screen can be seen in Figure 5-3 on page 339. Without the loopback adapter configured, the WebSphere Voice Server machine was not configured correctly to handle MRCP requests. In this case, the Cisco router terminated the call and played the default Cisco warning message to the caller.

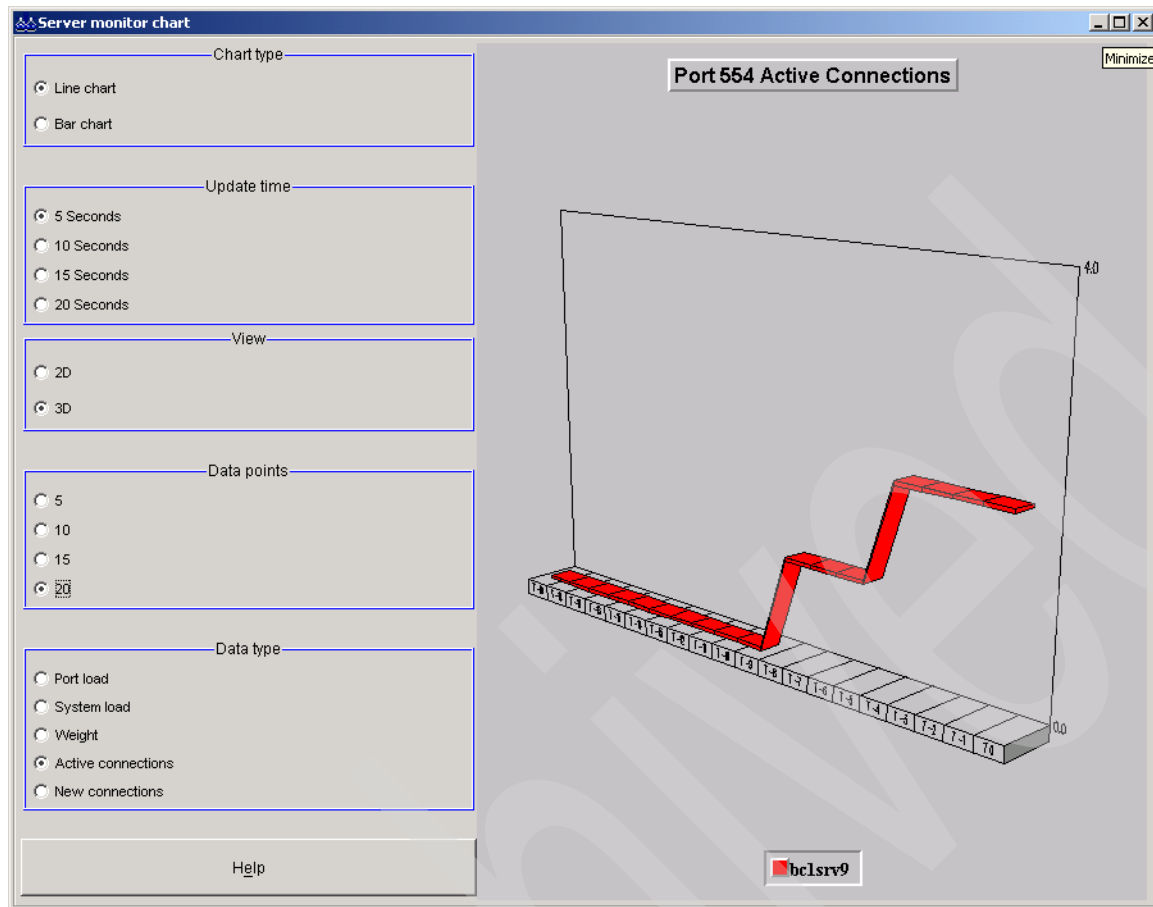


Figure 5-3 Active connection monitor

**Important:** In the ITSO lab even though a DNS server was running, and host name resolving for servers worked, the Load Balancer failed. To overcome this issue, each of the WebSphere Voice Server servers and the load balancer had the host name and IP address of each other in the hosts file. As soon as this was done, the system worked.

### 5.1.4 Network traffic analyzing

During our residency, we used certain troubleshooting tools. WebSphere Voice Server includes a variety of log and tracing functions. Several of them were used to diagnose problems. Refer to the WebSphere Voice Server Information Center and the *IBM WebSphere Voice Server for Multiplatforms V5.1.1/V5.1.2 Handbook, SG24-6447* for more information.

During the configuration of the Load Balancer, an issue appeared where calls were not flowing through the VoiceXML application. Load Balancer received the inbound MRCP request and then passed it to an available WebSphere Voice Server. The initial TTS function worked, but when the ASR engine was invoked, the call failed.

The WebSphere Voice Server trace function was used. This is configured in the troubleshooting screen, accessed from the Administrator console. Figure 5-4 on page 340 shows the actual trace option screen. The following trace options were used:

```
RECOGNIZER=entryExit=enabled,event=enabled:ASRENG=entryExit=enabled,event=enabled:ASRAPI=entryExit=enabled,event=enabled:RTSPBridge=entryExit=enabled,event=enabled
```



**Note:** Note that RTSPBridge was enabled and several ASR trace functions.

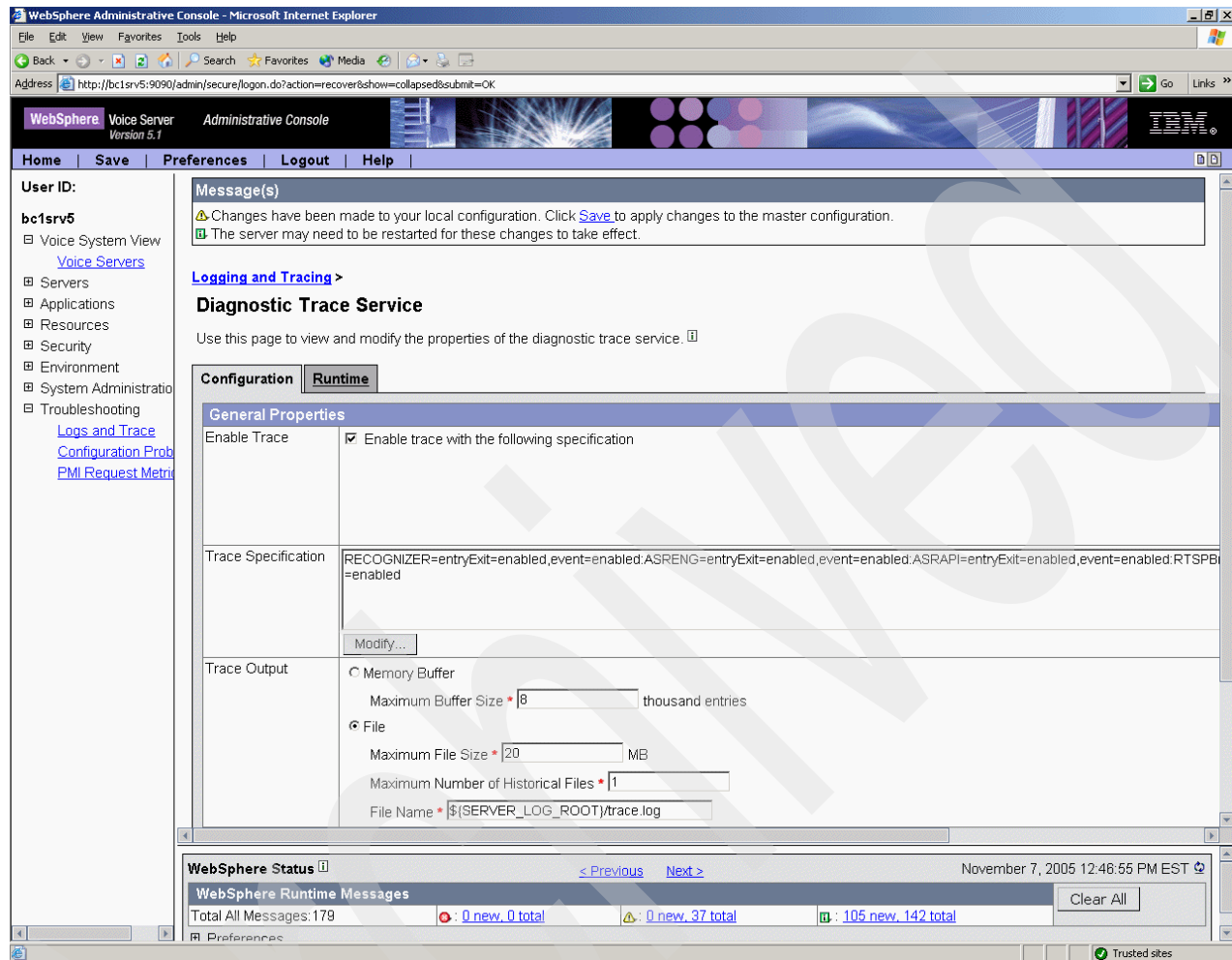


Figure 5-4 WebSphere Voice Server trace option screen

A call was made to reproduce the problem. This generated new additional WebSphere Voice Server messages. These can be viewed in the Runtime Events log. Figure 5-5 on page 341 has two messages that needed investigating.

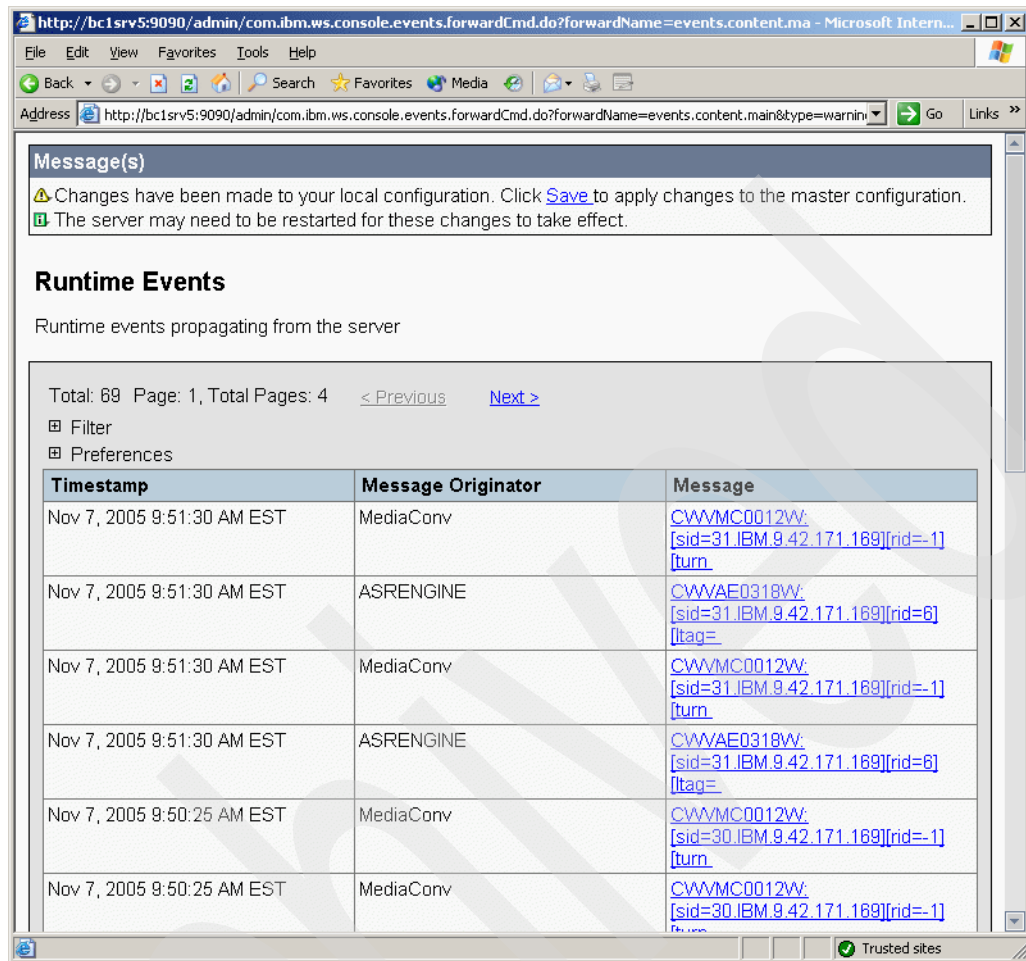


Figure 5-5 Runtime Events log after enabling tracing

The two top errors were reviewed. CWVMC0012W indicates an ASR error. No RTP packets were received by the WebSphere Voice Server. The full error description and the user action can be seen in Figure 5-6 on page 342.

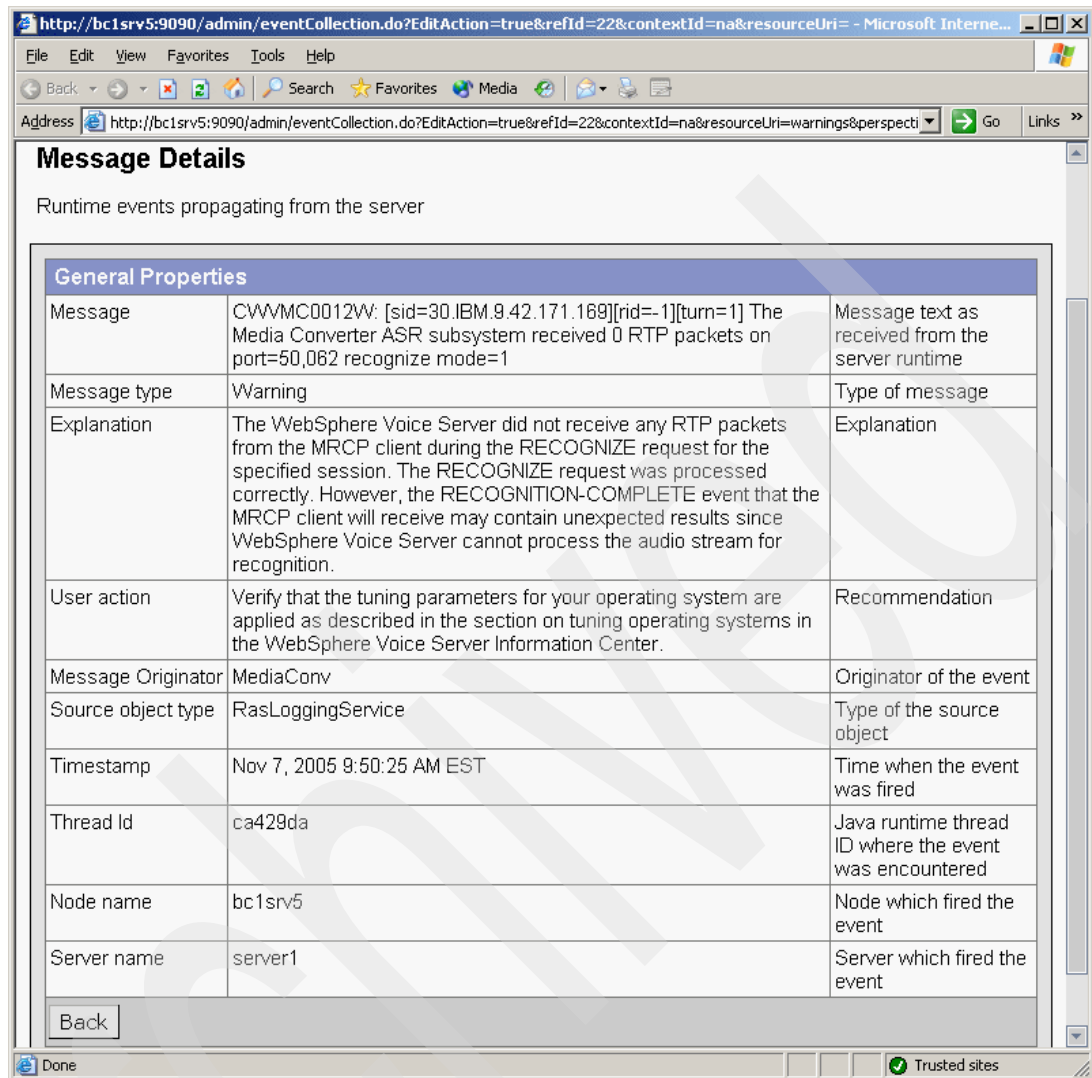


Figure 5-6 CWVMC0012W ASR error

The second error indicates the WebSphere Voice Server ASR received no audio input. After a ten second wait it terminated the function. The user action field suggests possible causes. The error CWVAE0318W and its full description can be seen in Figure 5-7 on page 343.

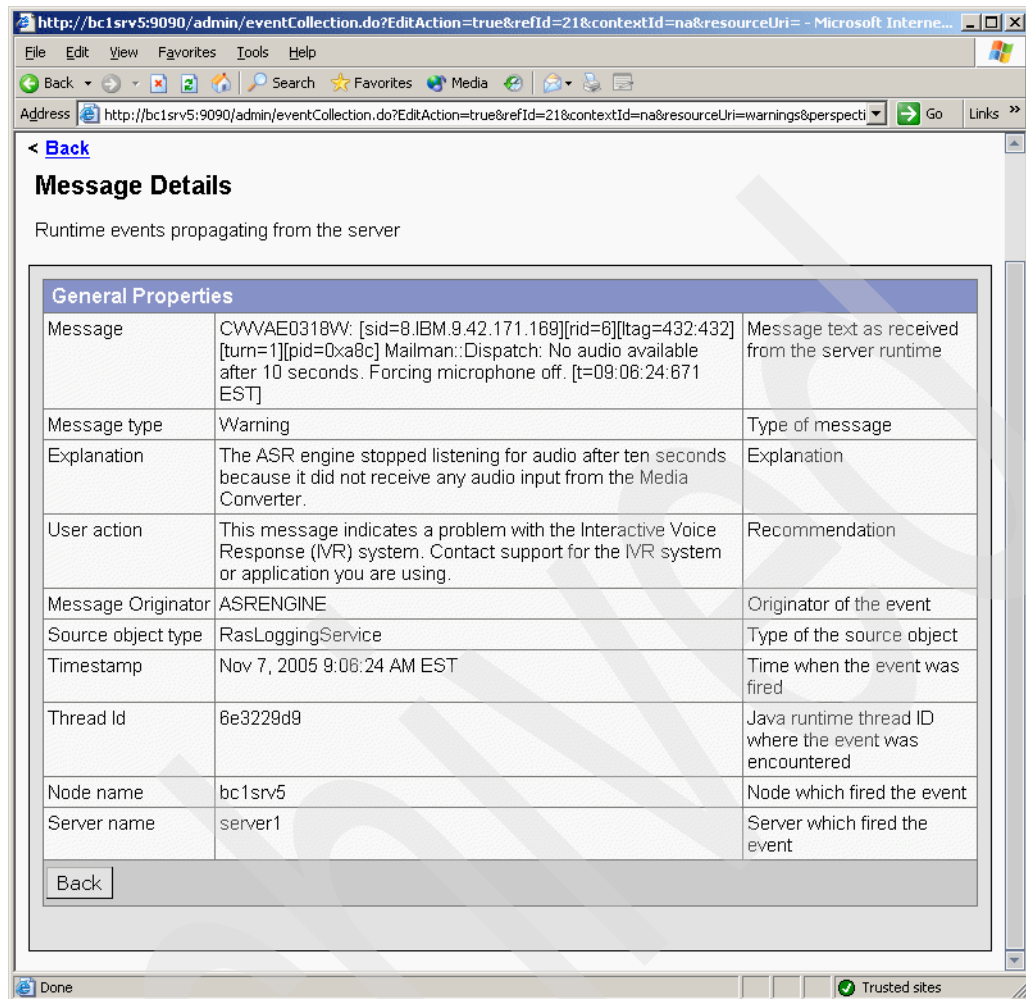


Figure 5-7 CWVAE0318W error

To assist us further, the Ethereal network protocol analyzer was used. This application is freely available from:

<http://www.ethereal.com>

The tool allowed us to analyze network traffic between the Cisco Router, WebSphere Load Balancer V5.1.1.41 and WebSphere Voice Server V5.1.3.

The Ethereal protocol analyzer was installed on the WebSphere Voice Server server. This server belonged to the Cluster 9.42.171.89 managed by the Load Balancer. A call was placed and a trace captured. Example 5-5 shows the analyzed trace and the specific error highlighted. The LB was reconfigured and the system worked then worked correctly.

Example 5-5 Ethereal trace

7	0.138825	9.42.171.89	9.42.171.120	RTSP	Reply: RTSP/1.0 405 Method Not Allowed
---	----------	-------------	--------------	------	--

## 5.2 Troubleshooting a Cisco Gateway problem

In developing a test script for the ICM Integrated application, we experienced the following problem:

- We made a test call from the POTS phone through the Gordon Kapes simulator to extension 7774001 - which was defined as the entry point for this test application. We heard a click, a few seconds of silence, then another click, then dial tone.

The first step was to determine how far the call got. Did it reach ICM? Did it reach the CVP Call Control Server? Did it even reach the Gateway? Did it get transferred to the VRU side for VoiceXML processing?

Starting from the top, we examined the ICM routing script. Fortunately we had just put the script into monitor mode and reset its counters, so this test call was the only call it would show. The routing script is shown in Figure 5-8.

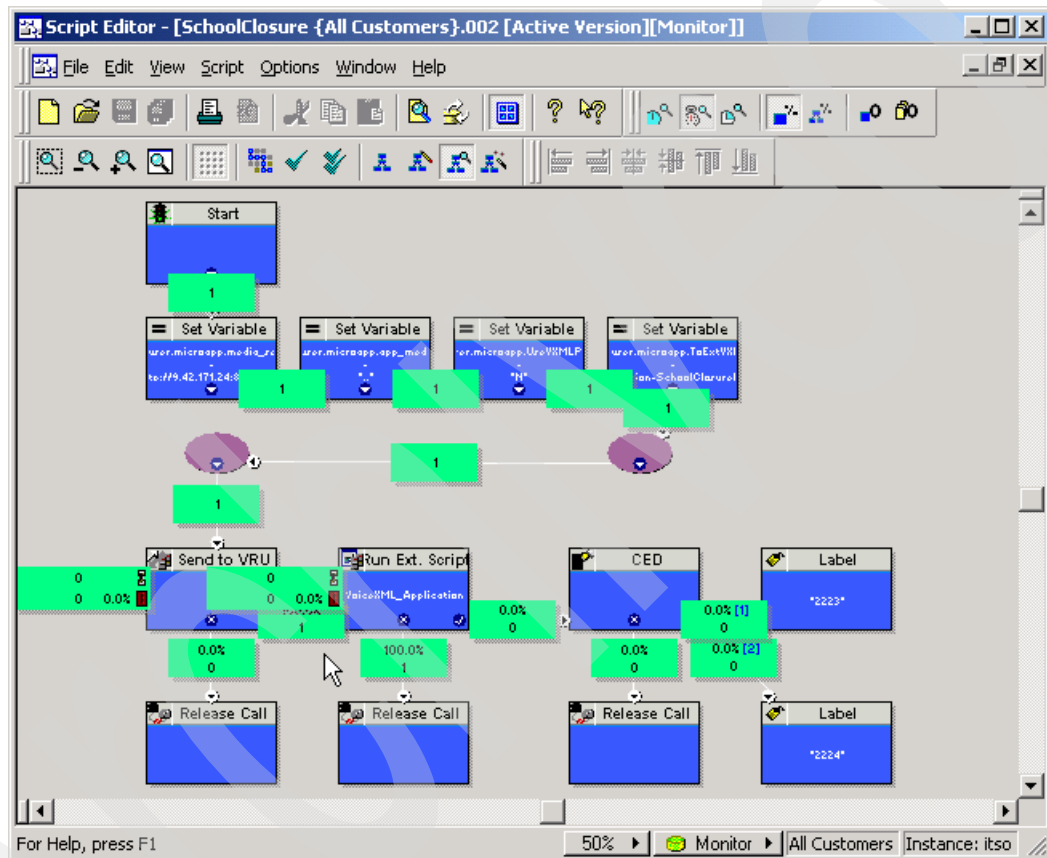


Figure 5-8 ICM Routing Script in Monitoring Mode

Notice that each path displays a green box indicating how many calls traversed that path. We looked for the following information:

- Did the call enter the script at all?
- Did the Send To VRU fail?
- Did any of the Run Ext. Script nodes fail?

If the call did not enter the script at all, we would see zero calls coming out of the start node, and we would look to the CVP Call Control Server and the Ingress Gateway for clues. In our case, we see that the call clearly entered the ICM routing script, so it is reaching ICM.

If the call failed out of the Send To VRU node, we would see a 1 on the failure branch (the *X* branch) from that node. That would indicate an error in the process of transferring the call to the VRU side, and we would look at the CVP Call Control Server, the Gatekeeper, and the dial-peers on the VoiceXML Gateway. However, that transfer appeared to succeed as well.

What we found was that the call failed in trying to execute the VoiceXML Server application.

Executing a VoiceXML Server application involves several components: the CVP Application Server (on the CVP Call Control Server machine), the VoiceXML Gateway, the VoiceXML Server itself, the Media Server, and the ASR/TTS server. The first component upstream from ICM is the CVP Application Server. We checked those logs by starting CVP App Admin at the following URL:

<http://9.42.171.150/appadmin>

From the main screen, we clicked **Engine**, then **Log Files**, and then on the only log file which appeared, **CiscoISN\_02.log**. Opening the log file, we found the following error message:

```
743: Nov 09 10:44:22.359 EST %ISN-SS_TEL-3-ERROR:CVP VXML Server encountered a Bad-Fetch
Error for http://9.42.171.24:8080/CVP/en-us/./Server?application=SchoolClosureInfo
call:5599EA31.506811DA.800E0006.53454EE6
```

Apparently, the VoiceXML Gateway had trouble fetching the VoiceXML Server application's start page. This narrows the issue down to those two devices.

To test whether the problem is with the VoiceXML Server, we entered the above URL directly into our Web browser's address bar. The result is shown in Figure 5-9 on page 345.

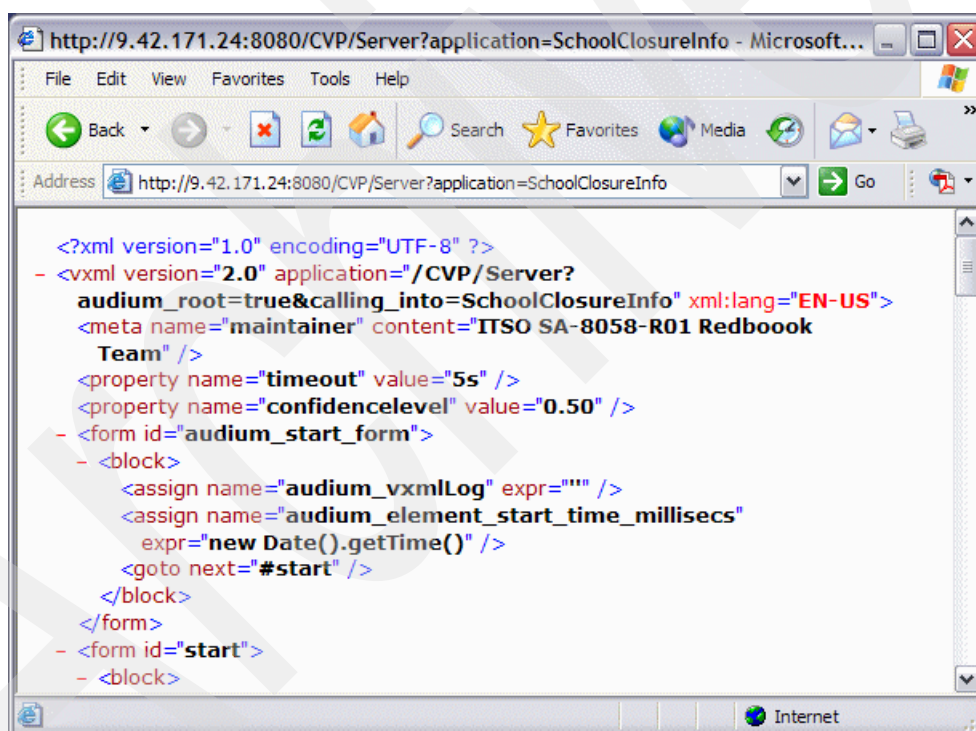


Figure 5-9 Results from fetching a VoiceXML Server page into a web browser

Clearly, the VoiceXML Server is having no problem with its application, so we now focus on the VoiceXML Gateway.



On the gateway, we type **show log** to see whether any errors appeared in the log file. Luck was with us; we found the following message:

```
%MEM_MGR-3-MEM_MGR_EXCEED_MAX: memory pool exceeds maximum (65536 bytes) allowed.
```

To learn what this message means, we went to [www.cisco.com](http://www.cisco.com) and logged in. Logging in is necessary in order to access the most useful information. Registration is free. There we performed the following steps:

1. On the right side of the screen, click **Advanced Search**.
2. In the Advanced Search screen, select **Technical Support & Documentation**, and paste the error code **MEM\_MGR-3-MEM\_MGR\_EXCEED\_MAX** into the Keywords field.
3. Click **Search**.

This unfortunately yielded no results. So we repeated the process specifying only **MEM\_MGR\_EXCEED\_MAX**. This yielded one match, a reference to the IOS version 12.3T System Message Guide.

Clicking that link took us to a large document containing many error messages. But we were able search within that document for the exact error code and found this explanation:

```
An attempt was made to allocate a buffer from a memory pool that has reached its maximum limit. The effected memory pool is specified in the message text.
```

The recommended action led us to the Cisco Bug Toolkit:

[http://www.cisco.com/cgi-bin/Support/Bugtool/launch\\_bugtool.pl](http://www.cisco.com/cgi-bin/Support/Bugtool/launch_bugtool.pl)

We launched the Bug Toolkit, and clicked **Next** under Search for Cisco IOS-related bugs by release.

On the bug search page, we entered the same error code in the Enter Keywords field, selected all the check boxes under Bug Status Group, and clicked **Next**.

Our efforts were rewarded when the Bug Toolkit turned up a single bug, number CSCeg26794. However, clicking that link produced a message that the bug has been Closed, and its details are only visible to Cisco employees.

As luck would have it, one of our Redpaper team members was a Cisco employee! He looked up the bug in Cisco's internal database and found that calls which remain active for a very long time end up using a large amount of memory. That memory is only released when the call terminates. If you do not happen to have a Cisco employee on your team, you can obtain the same information by calling Cisco TAC.

This information reminded us that a day earlier, due to some configuration errors, we had indeed had calls active for a very long time.

Armed with this explanation, we felt we could safely chalk this one up to lab experience, reboot the gateway, and try again. This time, the call succeeded.



# Cisco Router Configurations

This appendix shows the actual, full gateway configurations for the three IOS devices in our lab setup. The significant items have already been excerpted and described elsewhere in the document.

The following devices are included:

- ▶ Cisco 1751-V Modular Access Router Configuration
- ▶ Cisco AS5400HPX Universal Gateway Configuration
- ▶ Cisco 3660 Gatekeeper Configuration

An easy way to obtain a listing such as those shown here is to capture it using a terminal emulator such as HyperTerminal. Once you have the terminal emulator configured to capture output, type **term length 0** (to disable the more? message after every twenty lines), then **show running-config**, then **term length 20**. You will need to be in *enable* mode in order to view the gateway configuration.

**Note:** Avoid using the Microsoft command prompt window Telnet client because it has demonstrated difficulty in keeping up with the data rate and often drops lines. Missing lines of information in captured configuration files or debug listings can make the difference between being able to resolve an issue or not. HyperTerminal or another commercially available terminal emulator is recommended.

## A.1 Cisco 1751-V Modular Access Router Configuration

The Cisco 1751-V Modular Access Router configuration is shown in Example A-1. A directory listing of the flash memory is shown in Example A-2 on page 351.

*Example: A-1 Cisco 1751-V Modular Access Router Running-Config*

---

```
Current configuration : 4129 bytes
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
```



```

service SchoolClosureInfo flash:/CVPSelfService.tcl
  param CVPBackupVXMLServer 9.42.171.24
  paramspace english index 0
  paramspace english language en
  paramspace english location flash:
  param CVPSelfService-port 8080
  param CVPSelfService-app SchoolClosureInfo
  paramspace english prefix en
  param CVPPrimaryVXMLServer 9.42.171.24
!
service DTMFTest1 flash:CVPSelfService.tcl
  param CVPSelfService-app DTMFTest1
  param CVPSelfService-port 8080
  paramspace english language en
  paramspace english index 0
  param CVPPrimaryVXMLServer 9.42.171.24
  paramspace english location flash:
  paramspace english prefix en
  param CVPBackupVXMLServer 9.42.171.24
!
service CVPSelfService flash:CVPSelfServiceBootstrap.vxml
  paramspace english language en
  paramspace english index 0
  paramspace english location flash:
  paramspace english prefix en
!
service HelloWorld flash:/CVPSelfService.tcl
  param CVPSelfService-app HelloWorld
  param CVPSelfService-port 8080
  paramspace english language en
  paramspace english index 0
  param CVPPrimaryVXMLServer 9.42.171.24
  paramspace english location flash:
  paramspace english prefix en
  param CVPBackupVXMLServer 9.42.171.24
!
!
rtsp client timeout connect 10
rtsp client timeout message 10
mrsp client timeout connect 10
mrsp client timeout message 10
mrsp client rtpsetup enable
!
!
!
!
controller T1 1/0
  framing esf
  linecode b8zs
  pri-group timeslots 1-24
!
!
!
interface FastEthernet0/0
  ip address 9.42.171.131 255.255.255.0
  no ip mroute-cache
  speed 100
  full-duplex
  no cdp enable
!

```

```

interface Serial0/0
  no ip address
  service-module t1 cablelength short 110ft
  !
interface Serial1/0:23
  no ip address
  no logging event link-status
  isdn switch-type primary-ni
  isdn incoming-voice data
  no cdp enable
  !
interface Group-Async0
  physical-layer async
  no ip address
  no ip mroute-cache
  no group-range
  !
ip classless
ip route 0.0.0.0 0.0.0.0 9.42.171.3
!
no ip http server
no ip http secure-server
!
logging trap debugging
!
control-plane
!
!
!
voice-port 1/0:23
!
!
!
!
!
dial-peer cor custom
!
!
!
dial-peer voice 1001 voip
  service dtmf-test1
  incoming called-number 1001
  dtmf-relay rtp-nte h245-signal h245-alphanumeric
  codec g711ulaw
  no vad
!
dial-peer voice 10011 pots
  service dtmf-test1
  incoming called-number 1001
  direct-inward-dial
!
dial-peer voice 2001 voip
  service dtmf-test1
  incoming called-number 2001
  dtmf-relay rtp-nte h245-signal h245-alphanumeric
  codec g711ulaw
  no vad
!
dial-peer voice 20011 pots
  service dtmf-test1

```

```

incoming called-number 7772001
direct-inward-dial
!
dial-peer voice 69691 pots
service helloworld
incoming called-number 7776969
direct-inward-dial
!
dial-peer voice 9999 voip
service schoolclosureinfo
incoming called-number 9999
dtmf-relay rtp-nte h245-signal h245-alphanumeric
codec g711ulaw
no vad
!
dial-peer voice 6969 voip
service helloworld
incoming called-number 6969
dtmf-relay rtp-nte h245-signal h245-alphanumeric
codec g711ulaw
no vad
!
gateway
timer receive-rtp 1200
!
!
line con 0
line aux 0
line vty 0 4
exec-timeout 0 0
password password
login
!
end

```

---

*Example: A-2 Cisco 1751 Flash Memory Listing*

---

```

Router-1751#dir
Directory of flash:/

   1  -rw-     17882588                <no date>  c1700-ipvoicek9-mz.124-3a.bin
   6  -rw-         6367                <no date>  CVPSelfService.tcl
   8  -rw-     52062                  <no date>  error.wav
   9  -rw-         4830                <no date>  CVPSelfServiceBootstrap.vxm1

33292284 bytes total (15331916 bytes free)
Router-1751#

```

---

## A.2 Cisco AS5400HPX Universal Gateway Configuration

The Cisco AS5400HPX Universal Gateway configuration is shown in Example A-3. A directory listing of the flash memory is shown in Example A-4 on page 362.

---

*Example: A-3 Cisco AS5400HPX Universal Gateway Running-Config*

---

```

Current configuration : 10857 bytes
!
version 12.4

```

```

service config
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname AS5400
!
boot-start-marker
boot system flash c5400-is-mz.124-3a.bin
no boot startup-test
boot-end-marker
!
logging console critical
enable secret 5 $1$LmbD$w75nrg/ApWL9vS9Q/124C.
enable password password
!
!
!
resource-pool disable
no aaa new-model
!
resource policy
!
spe default-firmware spe-firmware-1
ip subnet-zero
!
!
ip cef
ip host asr-en-us 9.42.171.89
ip host asr-en-us-backup 9.42.171.89
ip host isn-vxml 9.42.171.150
ip host isn-vxml-backup 9.42.171.150
ip host tts-en-us 9.42.171.89
ip host tts-en-us-backup 9.42.171.89
ip host mediaserver 9.42.171.24
ip host mediaserver-backup 9.42.171.24
!
!
isdn switch-type primary-ni
!
voice call debug full-guid
voice call carrier capacity active
voice rtp send-recv
!
voice service voip
fax protocol t38 ls-redundancy 0 hs-redundancy 0 fallback cisco
h323
!
!
!
!
!
!
!
!
!
!
voice translation-rule 1
rule 1 /987654/ //
!
!

```

```

voice translation-profile block
  translate called 1
!
!
ivr prompt memory 16384
ivr prompt streamed all
ivr asr-server rtsp://9.42.171.89/media/recognizer
ivr tts-server rtsp://9.42.171.89/media/synthesizer
!
application
  service SchoolClosureInfo flash:CVPSelfService.tcl
    param CVPBackupVXMLServer 9.42.171.94
    paramspace english index 0
    paramspace english language en
    param CVPSElfService-app SchoolConfirmTest
    paramspace english location flash:
    param CVPSElfService-port 8080
    param CVPSElfService-app SchoolConfirmTest
    paramspace english prefix en
    param CVPPPrimaryVXMLServer 9.42.171.24
  !
  service new-call flash:bootstrap.vxml
    paramspace english language en
    paramspace english index 0
    paramspace english location flash:
    paramspace english prefix en
  !
  service CVPSElfService flash:CVPSElfServiceBootstrap.vxml
    paramspace english index 0
    paramspace english language en
    paramspace english location flash:
    paramspace english prefix en
  !
  service DTMFTest1 flash:CVPSElfService.tcl
    paramspace english index 0
    paramspace english language en
    param CVPSElfService-port 8080
    param CVPSElfService-app DTMFTest1
    paramspace english location flash:
    param CVPPPrimaryVXMLServer 9.42.171.24
    paramspace english prefix en
    param CVPBackupVXMLServer 9.42.171.94
  !
  service SchoolClosureInfoV2 flash:CVPSElfService.tcl
    param CVPBackupVXMLServer 9.42.171.94
    paramspace english index 0
    paramspace english language en
    paramspace english location flash:
    param CVPSElfService-port 8080
    param CVPSElfService-app SchoolClosureInfoV2
    paramspace english prefix en
    param CVPPPrimaryVXMLServer 9.42.171.24
  !
  service HelloWorld flash:CVPSElfService.tcl
    param CVPSElfService-app HelloWorld
    param CVPSElfService-port 8080
    paramspace english language en
    paramspace english index 0
    param CVPPPrimaryVXMLServer 9.42.171.24
    paramspace english location flash:

```

```

paramspace english prefix en
param CVPBackupVXMLServer 9.42.171.94
!
service handoff flash:handoff.tcl
paramspace english language en
paramspace english index 0
paramspace english location flash:
paramspace english prefix en
!
service bootstrap flash:bootstrap.tcl
paramspace english language en
paramspace english index 0
paramspace english location flash:
paramspace english prefix en
!
!
rtsp client timeout connect 10
rtsp client timeout message 10
mrp client timeout connect 10
mrp client timeout message 10
mrp client rtpsetup enable
no http client connection persistent
!
!
!
!
controller T1 1/0
framing esf
linecode b8zs
pri-group timeslots 1-24
!
controller T1 1/1
framing esf
linecode b8zs
pri-group timeslots 1-24
!
!
interface FastEthernet0/0
ip address dhcp
no ip redirects
ip route-cache same-interface
no ip mroute-cache
duplex full
speed 100
no keepalive
no cdp enable
h323-gateway voip interface
h323-gateway voip id Gatekeeper-3660 ipaddr 9.42.171.186 1719
h323-gateway voip h323-id 9.42.171.128
h323-gateway voip tech-prefix 1#
!
interface FastEthernet0/1
no ip address
shutdown
duplex auto
speed auto
no cdp enable
!
interface Serial0/0
no ip address

```



```

shutdown
clock rate 2000000
!
interface Serial10/1
no ip address
clock rate 2000000
no cdp enable
!
interface Serial11/0:23
no ip address
isdn switch-type primary-ni
isdn incoming-voice data
no cdp enable
!
interface Serial11/1:23
no ip address
isdn switch-type primary-ni
isdn incoming-voice data
no cdp enable
!
interface Async3/00
no ip address
!
interface Async3/01
no ip address
!
interface Async3/02
no ip address
!
interface Async3/03
no ip address
!
interface Async3/04
no ip address
!
interface Async3/05
no ip address
!
interface Async3/06
no ip address
!
interface Async3/07
no ip address
!
interface Async3/08
no ip address
!
interface Async3/09
no ip address
!
interface Async3/10
no ip address
!
interface Async3/11
no ip address
!
interface Async3/12
no ip address
!
interface Async3/13

```

```
no ip address
!
interface Async3/14
no ip address
!
interface Async3/15
no ip address
!
interface Async3/16
no ip address
!
interface Async3/17
no ip address
!
interface Async3/18
no ip address
!
interface Async3/19
no ip address
!
interface Async3/20
no ip address
!
interface Async3/21
no ip address
!
interface Async3/22
no ip address
!
interface Async3/23
no ip address
!
interface Async3/24
no ip address
!
interface Async3/25
no ip address
!
interface Async3/26
no ip address
!
interface Async3/27
no ip address
!
interface Async3/28
no ip address
!
interface Async3/29
no ip address
!
interface Async3/30
no ip address
!
interface Async3/31
no ip address
!
interface Async3/32
no ip address
!
interface Async3/33
```

```
no ip address
!
interface Async3/34
no ip address
!
interface Async3/35
no ip address
!
interface Async3/36
no ip address
!
interface Async3/37
no ip address
!
interface Async3/38
no ip address
!
interface Async3/39
no ip address
!
interface Async3/40
no ip address
!
interface Async3/41
no ip address
!
interface Async3/42
no ip address
!
interface Async3/43
no ip address
!
interface Async3/44
no ip address
!
interface Async3/45
no ip address
!
interface Async3/46
no ip address
!
interface Async3/47
no ip address
!
interface Async3/48
no ip address
!
interface Async3/49
no ip address
!
interface Async3/50
no ip address
!
interface Async3/51
no ip address
!
interface Async3/52
no ip address
!
interface Async3/53
```

```
no ip address
!
interface Async3/54
no ip address
!
interface Async3/55
no ip address
!
interface Async3/56
no ip address
!
interface Async3/57
no ip address
!
interface Async3/58
no ip address
!
interface Async3/59
no ip address
!
interface Async3/60
no ip address
!
interface Async3/61
no ip address
!
interface Async3/62
no ip address
!
interface Async3/63
no ip address
!
interface Async3/64
no ip address
!
interface Async3/65
no ip address
!
interface Async3/66
no ip address
!
interface Async3/67
no ip address
!
interface Async3/68
no ip address
!
interface Async3/69
no ip address
!
interface Async3/70
no ip address
!
interface Async3/71
no ip address
!
interface Async3/72
no ip address
!
interface Async3/73
```

```
no ip address
!
interface Async3/74
no ip address
!
interface Async3/75
no ip address
!
interface Async3/76
no ip address
!
interface Async3/77
no ip address
!
interface Async3/78
no ip address
!
interface Async3/79
no ip address
!
interface Async3/80
no ip address
!
interface Async3/81
no ip address
!
interface Async3/82
no ip address
!
interface Async3/83
no ip address
!
interface Async3/84
no ip address
!
interface Async3/85
no ip address
!
interface Async3/86
no ip address
!
interface Async3/87
no ip address
!
interface Async3/88
no ip address
!
interface Async3/89
no ip address
!
interface Async3/90
no ip address
!
interface Async3/91
no ip address
!
interface Async3/92
no ip address
!
interface Async3/93
```

```

    no ip address
    !
interface Async3/94
    no ip address
    !
interface Async3/95
    no ip address
    !
interface Async3/96
    no ip address
    !
interface Async3/97
    no ip address
    !
interface Async3/98
    no ip address
    !
interface Async3/99
    no ip address
    !
interface Async3/100
    no ip address
    !
interface Async3/101
    no ip address
    !
interface Async3/102
    no ip address
    !
interface Async3/103
    no ip address
    !
interface Async3/104
    no ip address
    !
interface Async3/105
    no ip address
    !
interface Async3/106
    no ip address
    !
interface Async3/107
    no ip address
    !
interface Group-Async0
    no ip address
    no group-range
    !
    !
ip classless
ip route 0.0.0.0 0.0.0.0 9.42.171.3
no ip http server
    !
    !
    !
    !
control-plane
    !
    !

```

```

!
voice-port 1/0:D
!
voice-port 1/1:D
!
!
dial-peer cor custom
!
!
!
dial-peer voice 1001 voip
  translation-profile incoming block
  service dtmf-test1
  incoming called-number 1001
  dtmf-relay rtp-nte h245-signal h245-alphanumeric
  codec g711ulaw
  no vad
!
dial-peer voice 10011 pots
  service dtmf-test1
  incoming called-number 7771001
  direct-inward-dial
!
dial-peer voice 2001 voip
  translation-profile incoming block
  service dtmf-test1
  incoming called-number 2001
  dtmf-relay rtp-nte h245-signal h245-alphanumeric
  codec g711ulaw
  no vad
!
dial-peer voice 20011 pots
  service dtmf-test1
  incoming called-number 7772001
  direct-inward-dial
!
dial-peer voice 6969 voip
  translation-profile incoming block
  service helloworld
  incoming called-number 6969
  dtmf-relay rtp-nte h245-signal h245-alphanumeric
  codec g711ulaw
  no vad
!
dial-peer voice 69691 pots
  service helloworld
  incoming called-number 7776969
  direct-inward-dial
!
dial-peer voice 9999 voip
  translation-profile incoming block
  service schoolclosureinfo
  incoming called-number 9999
  dtmf-relay rtp-nte h245-signal h245-alphanumeric
  codec g711ulaw
  no vad
!
dial-peer voice 91919191 voip
  translation-profile incoming block
  service schoolclosureinfov2

```

```

incoming called-number 91919191
dtmf-relay rtp-nte h245-signal h245-alphanumeric
codec g711ulaw
no vad
!
dial-peer voice 987654 voip
translation-profile incoming block
incoming called-number 987654
!
dial-peer voice 7774 pots
incoming called-number 777....
direct-inward-dial
!
dial-peer voice 77740 voip
translation-profile incoming block
destination-pattern 777....
session target ras
tech-prefix 1#
dtmf-relay rtp-nte h245-signal h245-alphanumeric
codec g711ulaw
no vad
!
dial-peer voice 8887878 voip
translation-profile incoming block
service bootstrap
incoming called-number 8887878T
dtmf-relay rtp-nte h245-signal h245-alphanumeric
codec g711ulaw
no vad
!
!
gateway
timer receive-rtp 1200
!
ss7 mtp2-variant Bellcore 0
ss7 mtp2-variant Bellcore 1
ss7 mtp2-variant Bellcore 2
ss7 mtp2-variant Bellcore 3
!
line con 0
exec-timeout 0 0
line aux 0
line vty 0 4
password password
login
line 3/00 3/107
no flush-at-activation
modem InOut
!
scheduler allocate 10000 400
end

```

---

*Example: A-4 Cisco AS5400HPX Universal Gateway Flash Memory Contents*

---

```

AS5400#dir
Directory of flash:/

 1  -rw-   13847396  May 20 2003 20:26:44 +00:00  c5400-is-mz.123-1.5
 2  -rw-   17530928  Oct 31 2005 16:07:49 +00:00  c5400-is-mz.124-3a.bin
 3  -rw-      6367   Oct 31 2005 17:04:19 +00:00  CVPSelfService.tcl

```



4	-rw-	4830	Oct 31 2005 17:04:46 +00:00	CVPSelfServiceBootstrap.vxml
5	-rw-	52062	Oct 31 2005 17:05:14 +00:00	error.wav
6	-rw-	2846	Nov 02 2005 15:10:46 +00:00	bootstrap.tcl
7	-rw-	2171	Nov 02 2005 15:11:03 +00:00	bootstrap.vxml
8	-rw-	2446	Nov 02 2005 15:11:42 +00:00	en_0.wav
9	-rw-	2446	Nov 02 2005 15:11:51 +00:00	en_1.wav
10	-rw-	2446	Nov 02 2005 15:12:05 +00:00	en_2.wav
11	-rw-	2446	Nov 02 2005 15:12:16 +00:00	en_3.wav
12	-rw-	2446	Nov 02 2005 15:12:27 +00:00	en_4.wav
13	-rw-	2446	Nov 02 2005 15:12:38 +00:00	en_5.wav
14	-rw-	2446	Nov 02 2005 15:12:49 +00:00	en_6.wav
15	-rw-	2446	Nov 02 2005 15:12:58 +00:00	en_7.wav
16	-rw-	2446	Nov 02 2005 15:13:07 +00:00	en_8.wav
17	-rw-	2446	Nov 02 2005 15:13:18 +00:00	en_9.wav
18	-rw-	54522	Nov 02 2005 15:13:29 +00:00	en_error.wav
19	-rw-	126454	Nov 02 2005 15:13:43 +00:00	en_holdmusic.wav
20	-rw-	26582	Nov 02 2005 15:14:01 +00:00	en_pleasewait.wav
21	-rw-	2446	Nov 02 2005 15:14:39 +00:00	en_pound.wav
22	-rw-	2446	Nov 02 2005 15:14:52 +00:00	en_star.wav
23	-rw-	80058	Nov 02 2005 15:15:31 +00:00	fetchwait.wav
24	-rw-	1186	Nov 02 2005 15:15:51 +00:00	handoff.tcl
25	-rw-	557	Nov 02 2005 15:17:18 +00:00	recovery.vxml
26	-rw-	91417	Nov 02 2005 15:17:34 +00:00	survivability.tcl
27	-rw-	10857	Nov 08 2005 15:43:46 +00:00	running-config-bak-08-11-05

65536000 bytes total (33664908 bytes free)  
AS5400#

---

## A.3 Cisco 3660 Gatekeeper Configuration

The Cisco 3660 Gatekeeper configuration is shown in Example A-5. A directory listing of the flash memory is shown in Example A-6 on page 366.

*Example: A-5 Cisco 3660 Gatekeeper Running-Config*

---

```

Current configuration : 1859 bytes
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname Gatekeeper-3660
!
boot-start-marker
boot system flash c3660-ix-mz.124-3a.bin
boot-end-marker
!
logging buffered 1000000 debugging
no logging console
enable secret 5 $1$x5Vu$JCWgVKry1/SE.3x4V1Amm.
enable password password
!
no aaa new-model
!
resource policy
!
ip subnet-zero

```

```

!
!
ip cef
!
!
!
!
voice call carrier capacity active
!
voice service voip
!
!
!
!
!
!
!
!
!
!
interface FastEthernet0/0
 ip address 9.42.171.186 255.255.255.0
 no ip route-cache cef
 no ip route-cache
 speed 100
 full-duplex
!
interface FastEthernet0/1
 ip address dhcp
 no ip route-cache cef
 no ip route-cache
 shutdown
 speed 100
 full-duplex
!
interface FastEthernet1/0
 no ip address
 shutdown
 half-duplex
!
interface Serial2/0
 no ip address
 shutdown
 serial restart-delay 0
!
interface Serial2/1
 no ip address
 shutdown
 serial restart-delay 0
!
interface Serial2/2
 no ip address
 shutdown
 serial restart-delay 0
!
interface Serial2/3
 no ip address
 shutdown
 serial restart-delay 0

```

```
!  
interface Serial4/0  
  no ip address  
  shutdown  
  serial restart-delay 0  
!  
interface Serial4/1  
  no ip address  
  shutdown  
  serial restart-delay 0  
!  
interface Serial4/2  
  no ip address  
  shutdown  
  serial restart-delay 0  
!  
interface Serial4/3  
  no ip address  
  shutdown  
  serial restart-delay 0  
!  
no ip http server  
!  
ip classless  
!  
!  
!  
control-plane  
!  
!  
dial-peer cor custom  
!  
!  
!  
!  
gatekeeper  
  zone local Gatekeeper-3660 itso.ral.ibm.com 9.42.171.186  
  zone prefix Gatekeeper-3660 2... gw-priority 10 9.42.171.106  
  zone prefix Gatekeeper-3660 777* gw-priority 10 9.42.171.150  
  zone prefix Gatekeeper-3660 8887878* gw-priority 10 9.42.171.128  
  gw-type-prefix 1#* default-technology  
  no shutdown  
!  
!  
line con 0  
line aux 0  
line vty 0 4  
  password password  
  login  
!  
!  
end
```

---

*Example: A-6 Cisco 3660 Gatekeeper Flash Memory listing*

---

Gatekeeper-3660#dir  
Directory of flash:/

3	-rw-	12575528	<no date>	c3660-ix-mz.124-3a.bin
---	------	----------	-----------	------------------------

66584572 bytes total (10863036 bytes free)  
Gatekeeper-3660#

---

## Additional material

This Redpaper refers to additional material that can be downloaded from the Internet as described below.

### Locating the Web material

The Web material associated with this Redpaper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/REDP4025>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select the **Additional materials** and open the directory that corresponds with the Redpaper form number, REDP4025.

### Using the Web material

The additional Web material that accompanies this Redpaper includes the following files:

<i>File name</i>	<i>Description</i>
<b>REDP4025.zip</b>	Zipped Code Samples

### System requirements for downloading the Web material

The following system configuration is recommended:

<b>Hard disk space:</b>	2 MB
<b>Operating System:</b>	Windows
<b>Processor:</b>	1 GHz or higher
<b>Memory:</b>	1 GB

## How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

# Glossary

**ANI.** See Automatic Number Identification.

**ASR.** See Automatic Speech Recognition.

**Automatic Number Identification (ANI).** A service that tells the recipient of a telephone call the telephone number of the person making the call.

**Automatic Speech Recognition (ASR).** Speech recognition technologies allow computers equipped with a source of sound input, such as a telephone handset or microphone, to interpret human speech.

**barge-in.** A capability that allows callers to speak or enter their responses during the prompt.

**Call Control eXtensible Markup Language (CCXML).** is an XML standard designed to provide telephony support to VoiceXML. Where as VoiceXML is designed to provide a VUI interface to a voice browser, CCXML is designed to inform the voice browser how to handle the telephony control of the voice channel. The two languages are wholly separate and are not required by each other to be implemented.

**call flow.** The "path" that a caller is directed through when interacting with a voice response system.

**CCXML.** See Call Control eXtensible Markup Language.

**cluster.** A group of locally connected computers that work together as a unit.

**component.** A pre-defined segment of a voice-driven application.

**Computer Telephony Integration (CTI).** The name given to the merger of traditional telecommunications (PBX) equipment with computers and computer applications. The use of caller ID to retrieve customer information automatically from a database is an example of a CTI application.

**confidence Score.** A value that indicates how certain the recognizer is that the result is correct. The confidence value can be used by the application to decide whether to proceed without asking for confirmation of the data, whether to ask for confirmation of the data, or whether to ask the caller to repeat the data.

**configuration.** The arrangement of the software and hardware in a computer system or network.

**confirmation.** Short dialog flow that repeats a recognition result back to the caller and asking him or her to confirm that it is correct.

**CTI.** See Computer Telephony Integration

**database.** A collection of information stored in a computer in a systematic way, such that a computer program can consult it to answer questions.

**debug.** The process of locating and correcting errors in computer programs.

**default.** A particular setting or value for a variable that is assigned automatically by an operating system and remains in effect unless canceled or overridden by the operator.

**Dialed Number Identification Service (DNIS).** A telephone function that sends the dialed telephone number to the answering service.

**dialog box.** A secondary window in which you are asked to fill in information.

**dialog.** Interactions between the caller and the application are called dialogs.

**directory.** A type of file used to group and organize other files or directories. Also called a folder.

**DNIS.** See Dialed Number Identification Service.

**DTMF.** See Dual Tone Multi Frequency tones.

**Dual Tone Multi Frequency tones (DTMF).** The system used by touch-tone telephones. DTMF assigns a specific frequency (made up of two separate tones) to each key so that it can easily be identified by a microprocessor. This is basically the technology behind touch tone dialing.

**form.** A document (printed or electronic) with spaces in which to write or enter data.

**Graphical User Interface (GUI).** a user interface based on graphics (icons and pictures and menus) instead of text.

**GUI.** See Graphical User Interface.

**hardware.** The physical components of a computer system, such as keyboards, monitors, and media disk drives.

**Integrated Services Digital Network (ISDN).** An international standard for end-to-end digital transmission of voice, data and signaling.

**Interactive Voice Response (IVR).** A telecommunications system that uses prerecorded voice messages to present options to a user.

**interface.** The point of interaction or communication between a computer and any other entity, such as a printer or human operator.

**ISDN.** See Integrated Services Digital Network.

**IVR.** See Interactive Voice Response.

**J2EE.** See Java 2 Platform, Enterprise Edition.

**Java 2 Platform, Enterprise Edition (J2EE).** An environment for developing and deploying enterprise applications, defined by Sun Microsystems™ Inc. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multi-tiered, Web-based applications.

**Media Resource Control Protocol (MRCP).** The Media Resource Control Protocol created by the Internet Engineering Task Force (IETF). It is a communication protocol which allows speech servers to provide various speech services (such as speech recognition and speech synthesis) to its clients. Typically, this means the server software will be running on one computer and clients can send MRCP messages to the server over a network, usually on top of another protocol, such as RTSP or TCP.

**menu.** A graphical user interface element which allows the user to select one among several, presumably related, choices.

**module.** A separate and distinct unit of hardware or software that may be used as a component in a system.

**MRCP.** See Media Resource Control Protocol.

**Natural Language Speech Recognition (NLSR).** An advanced type of speech recognition. NLSR can recognize particular words and phrases spoken by the user.

**NLSR.** See Natural Language Speech Recognition.

**node.** A device connected to a network, such as a computer or router.

**parameter.** A value or object which is used to transfer information to or from subprograms. Input parameters can be thought of as being copied into the subprogram when it is called, and output parameters can be thought of as having their values copied back to the caller when the subprogram returns.

**PBX.** See Private Branch eXchange.

**platform.** A platform refers to a framework on which applications may be run.

**port.** A connection to a device that allows data or information to travel into and out of the device.

**predictive dialing.** Automated dialing feature in which the systems predicts from pre-set algorithms when an agent/operator will become free and makes the call in advance.

**Private Branch eXchange (PBX).** A private telephone switching system, usually located on a company's premises.

**prompt.** a message played to a caller that gives the caller a choice of selections in a menu and asks for a response. Compare to announcement.

**proxy server.** A server that receives requests intended for another server and that acts on the behalf of the client behalf (as the client proxy) to obtain the requested service. A proxy server is often used when the client and the server are incompatible for direct connection.

**PSTN.** See Public Switched Telephone Network.

**Public Switched Telephone Network (PSTN).** The traditional analog network that routes voice calls from one location to another.

**query.** A query is a form of questioning in SQL language to retrieve information from a database.

**RDBMS.** See Relational Database Management System.

**Real Time Streaming Protocol (RTSP).** A client/Server communication protocol that simplifies the distribution of multimedia contents on the Internet.

**Red Hat.** A software company in the business of assembling open source components for the Linux operating system and related programs into a distribution package.

**Relational Database Management System (RDBMS).** T type of DBMS in which the database is organized and accessed according to the relationships between data values.



**resilience.** The ability of a structure to sustain the impact of a business interruption and recover and resume its operations to continue to provide minimum services.

**round robin.** A simple mechanism used by DNS servers to share and distribute loads for network resources.

**RTSP.** See Real Time Streaming Protocol.

**scalability.** The ability to increase or decrease size or capability in cost-effective increments with minimal impact on the unit cost of business and the procurement of additional services.

**script.** The set of instructions for the target voice response system to follow during a transaction.

**servlet.** A server-side Java program that provides additional features to the server.

**software.** The set or sets of programs that instruct the computer hardware to perform a task or series of tasks.

**Speech Synthesis Markup Language (SSML).** A markup language to control speech synthesis and text processing defined by the World Wide Web Consortium (W3C).

**SSML.** See Speech Synthesis Markup Language.

**SUSE.** A privately owned company whose mission is to promote open source development and General Public License distribution and to be a Linux distribution provider. SUSE assembles open source components for the Linux operating system and related programs into a selection of distribution packages.

**Text-to-Speech (TTS).** An optional feature that lets an application play language speech directly from ASCII text by converting that text to synthesized speech. The text can be used for prompts or for text retrieved from a database or host, and can be spoken in an application with prerecorded speech.

**thin client.** A low-cost computing device that accesses applications and/or data from a central server over a network. Categories of thin clients include Windows-Based Terminals (WBT, which comprise the largest segment), X-Terminals, and Network Computers (NC).

**topology.** The physical layout of a network.

**troubleshooting.** The process of locating and correcting errors in computer programs.

**TTS.** See Text-to-Speech.

**turnkey.** A product or system that can be plugged in, turned on, and operated with little or no additional configuring.

**Uniform Resource Identifier (URI).** A URI is a formatted string used to locate via name or location a resource document such as a VoiceXML document.

**URI.** See Uniform Resource Identifier

**Voice eXtensible Markup Language (VoiceXML).** The W3C's standard XML format for specifying interactive voice dialogues between a human and a computer. It is fully analogous to HTML, and brings the same advantages of web application development and deployment to voice applications that HTML brings to visual applications.

**VoiceXML.** See Voice eXtensible Markup Language.

**WAR.** See Web ARchive.

**Web ARchive (WAR).** A Web ARchive file is the new J2EE standard for packaging Java Servlet based application, Java Server Pages and associated files and deploying them.

Archived

# Abbreviations and acronyms

<b>ACD</b>	Automatic Call Distribution
<b>ANI</b>	Automatic Number Identification
<b>ASR</b>	Automatic Speech Recognition
<b>CCXML</b>	Call Control Extensible Markup Language
<b>CSS</b>	Content Service Switch
<b>CTI</b>	Computer Telephony Integration
<b>DNIS</b>	Dialed Number Identification Service
<b>DSP</b>	Digital Signal Processor
<b>DTMF</b>	Dual Tone Multi Frequency
<b>GUI</b>	Graphical User Interface
<b>IBM</b>	International Business Machines Corporation
<b>ICM</b>	Intelligent Call Management
<b>IPCC</b>	Internet Protocol Contact Center
<b>ISDN</b>	Integrated Services Digital Network
<b>ITSO</b>	International Technical Support Organization
<b>J2EE</b>	Java 2 Platform, Enterprise Edition
<b>MRCP</b>	Media Resource Control Protocol
<b>NLSR</b>	Natural Language Speech Recognition
<b>PBX</b>	Private Branch Exchange
<b>PSTN</b>	Public Switched Telephone Network
<b>RDBMS</b>	Relational Database Management System
<b>RTSP</b>	Real Time Streaming Protocol
<b>SSML</b>	Speech Synthesis Markup Language
<b>TDM</b>	Time Division Multiplexing
<b>TTS</b>	Text to Speech
<b>URI</b>	Uniform Resource Identifier
<b>VoiceXML</b>	Voice Extensible Markup Language
<b>VRU</b>	Voice Response Unit

Archived

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this Redpaper.

## IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 376. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *WebSphere Voice Server for Multiplatforms V5.1.1/V5.1.2 Handbook*, SG24-6447

## Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ Microsoft Windows Server 2003 TechCenter:  
<http://www.microsoft.com/technet/prodtechnol/windowsserver2003/default.msp>
- ▶ Red Hat Enterprise Linux server:  
<http://www.redhat.com/software/rhel>
- ▶ Softel Home Page:  
<http://www.softel.com>
- ▶ SUSE LINUX Enterprise Server:  
<http://www.novell.com/products/linuxenterpriseserver>
- ▶ World Wide Web Consortium (W3C) Home Page:  
<http://www.w3c.org>
- ▶ W3C recommendation of Speech Recognition Grammar Specification (SRGS) V1.0:  
<http://www.w3.org/TR/2004/REC-speech-grammar-20040316>
- ▶ W3C recommendation of Speech Synthesis Markup Language (SSML) V1.0:  
<http://www.w3.org/TR/2004/REC-speech-synthesis-20040907>
- ▶ W3C working draft of Semantic Interpretation for Speech Recognition (SISR) :  
<http://www.w3.org/TR/2003/WD-semantic-interpretation-20030401>
- ▶ W3C recommendation for Voice Extensible Markup Language (VoiceXML) V2.0:  
<http://www.w3.org/TR/voicexml20>
- ▶ WebSphere Application Server Zone:  
<http://www.ibm.com/developerworks/websphere/zones/was>
- ▶ WebSphere Application Server V5.1.x Information Center:  
[http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.websphere.bas.e.doc/info/welcome\\_base.html](http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.websphere.bas.e.doc/info/welcome_base.html)
- ▶ WebSphere Application Server Network Deployment V5.1.x Information Center:  
[http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/welcome\\_nd.html](http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/welcome_nd.html)

- ▶ WebSphere Application Server Base and Network Deployment editions recommended updates:  
<http://www.ibm.com/support/docview.wss?rs=180&uid=swg27004980>
- ▶ WebSphere Edge Server for Multiplatforms Network Dispatcher (Load Balancer) Administration Guide:  
<ftp://ftp.software.ibm.com/software/websphere/edgeserver/info/doc/v20/en/NDguide.pdf>
- ▶ WebSphere Voice family product documentation library:  
<http://www.ibm.com/developerworks/websphere/zones/voice/proddoc.html#wvs>
- ▶ WebSphere Voice Server Education and Class Information:  
<http://www.ibm.com/developerworks/websphere/education/enabement>
- ▶ WebSphere Voice Server for Multiplatforms V5.1.x Information Center:  
<http://publib.boulder.ibm.com/infocenter/pvcvoice/51x/index.jsp>
- ▶ WebSphere Voice Server Support Page:  
[http://www-306.ibm.com/software/pervasive/voice\\_server/support](http://www-306.ibm.com/software/pervasive/voice_server/support)
- ▶ WebSphere Voice Server System Requirements:  
[http://www.ibm.com/software/pervasive/voice\\_server/system\\_requirements](http://www.ibm.com/software/pervasive/voice_server/system_requirements)
- ▶ WebSphere Voice Server IVR and gateway compatibility web page:  
[http://www.ibm.com/software/pervasive/voice\\_server/ivrgateway.html](http://www.ibm.com/software/pervasive/voice_server/ivrgateway.html)
- ▶ WebSphere Voice Toolkit:  
[http://www.ibm.com/software/pervasive/voice\\_toolkit](http://www.ibm.com/software/pervasive/voice_toolkit)
- ▶ WebSphere Voice Zone:  
<http://www.ibm.com/developerworks/websphere/zones/voice>

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)





# IBM WebSphere Voice Server V5.1.2/V5.1.3 and Cisco Customer Voice Portal V3.1: An Interoperability Guide



**Using MRCP for  
interoperability**

**Developing VoiceXML  
applications**

**Debugging across  
platforms**

We wrote this IBM Redpaper to be an interoperability guide for integrating WebSphere Voice Server for Multiplatforms V5.1.2/V5.1.3 and Cisco Customer Voice Portal V3.1. We give you a broad understanding of how open standards like VoiceXML, Media Resource Control Protocol (MRCP), Speech Recognition Grammar Specification (SRGS) and Speech Synthesis Markup Language (SSML) are used.

In this paper, we consider best practices as applied to Voice User Interface (VUI) design. We implement and deploy a simple voice-enabled application using Cisco Customer Voice Portal that will interoperate with WebSphere Voice Server V5.1.2/V5.1.3.

We demonstrate how to develop, test, and deploy a simple voice-enabled application using the VoiceXML markup language. We also demonstrate the use of Automatic Speech Recognition (ASR) and Text to Speech (TTS) voice technologies through examples.

With this paper, you can tailor and configure WebSphere Voice Server and Cisco Customer Voice Portal to interoperate in simple and complex topologies.

We assume you have a basic knowledge of Interactive Voice Response (IVR) systems and voice-enablement of applications using VoiceXML, Cisco Customer Voice Portal, WebSphere Voice Server, and WebSphere Application Server.

**INTERNATIONAL  
TECHNICAL  
SUPPORT  
ORGANIZATION**

**BUILDING TECHNICAL  
INFORMATION BASED ON  
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:  
[ibm.com/redbooks](http://ibm.com/redbooks)**