# Training The Experts
# Call Studio Application Development for CVP VoiceXML Server (CVPD) v8.0
# Student Guide

TECH 2000 INCORPORATED

CISCO

# Cisco® CVPD

## Call Studio Application Development for CVP VoiceXML Server

TrainingTheExperts JGraves@TrainingTheExperts.com
Training the Experts is a reseller of Tech 2000 Inc., a Cisco® Learning Solutions Partner.

TrainingTheExperts JGraves@TrainingTheExperts.com

# Training the Experts

***Expert Training in VoiceXML, Speech and IVR Programming***

## Table of Contents




TrainingTheExperts JGraves@TrainingTheExperts.com

Training the Experts is a reseller of Tech 2000 Inc., a Cisco® Learning Solutions Partner.

- The Web Services Element

8. Working with Multiple Studio Applications: Studio Subdialogs and Application Transfers
   - The Subdialog Invoke Element to use a Studio Application as a Subroutine
   - Writing the Subdialog: Receiving and Returning Data
   - Accessing Returned Data
   - Application Transfer to transfer control to another Studio Application

9. Miscellaneous
   - Hotlinks – Enabling Global Key presses And Utterances
   - Hotevents – Catching VXML Events
   - Error Element – Handling Unrecoverable Errors
   - Record (with Confirm) – Taking a Recording From the Caller
   - FTP Element
   - MultiLanguage Applications - Using Application Modifier Element
   - Request ICM Label – Interfacing to ICM Within the Studio Call Flow

10. Working with Custom Java Components
    - Introduction to the CVP Java and XML APIs
    - Deploying Custom Java Components
    - Creating a Java Project in Eclipse
    - Implementing Custom Components
    - Standard Actions
    - Standard Decisions
    - Configurable Action and Decision Elements
    - On Start of Call and On End of Call
    - Dynamic Configurations for Dynamic Menus
    - Say it Smart Plugins

11. Courtesy Callback
    - Courtesy Callback Overview
    - ICM Configuration and Demo Courtesy Callback Script
    - Studio and VXML Server Courtesy Callback Applications
    - Reporting Server Tables for Courtesy Callback
    - Media Server Configuration for Courtesy Callback

12. Operations Console and Reporting Server
    - Using the Operations Console to Deploy a VXMLServer Application
    - Configuring the Reporting Server
    - Configuring VXMLServer information to Send to the Reporting Server
    - Working with Reporting Server Tables

13. Call Studio Documenter Printout
    - Invoking the Studio Documenter to Print an Application
    - Sample Printout

# Chapter 1

# CVP Introduction and Call Flow

- CVP Overview
- CVP Feature Set
- CVP Hardware Components
- CVP Call Flow
- Configuring ICM to Invoke VXML Server Applications

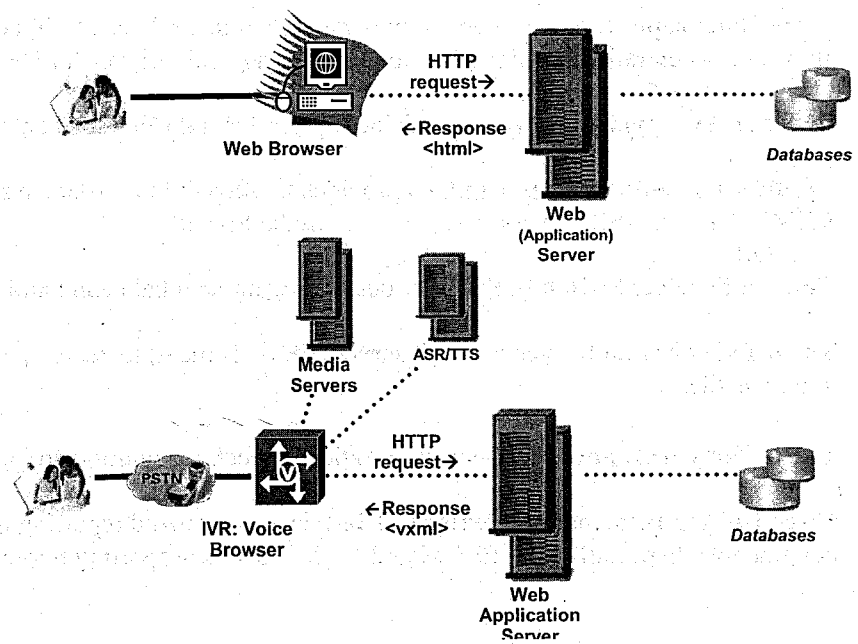
## CVP and VoiceXML

1. **Unified CVP** is a solution that uses Voice over IP to provide carrier-class IVR and IP switching services, including:
   A. IVR Services
   B. Queuing treatment
   C. Integration with Cisco Unified Contact Center and ICM to control CVP VoIP switching and IVR services
   D. IP-based call switching to route calls between voice gateways and IP endpoints
   E. CVP Operations Console management portal for administrative and configuration tasks for CVP solution components
   F. VRU reporting with a centralized reporting database for real-time and historical reporting

2. The IVR portion of CVP uses **VoiceXML**, a standards-based language, to provide web server type development and content delivery for customer self-service.

3. In direct analogy to the internet's use of **HTML** documents created dynamically by remote **web servers** and interpreted by **web browsers**; the CVP solution uses **VoiceXML** documents created dynamically by remote application servers (Tomcat or WebSphere) and interpreted by a **voice browser.**

4. The Cisco **voice browser** resides on the **Cisco VoiceXML Gateway** ("IVR" below). It interprets VoiceXML documents retrieved from the application server (VXML Server or Call Server's IVR Service) using HTTP-submit; it uses HTTP to retrieve, cache, and play audio files from the Media Server; uses MRCP to interact with speech recognition and text-to-speech servers; natively collects DTMF tones; and uses HTTP-submit to return collected data back to the application server to continue the self-service application.



## Unified CVP Components

1. Unified CVP is both a product and a solution.

2. The **Unified CVP product** consists of (discussed in more detail on the next page):
   A. **CVP Call Server** – interface between SIP or H323 endpoints and ICM
   B. **CVP VXML Server** and **Call Studio** – create and run self-service IVR applications
   C. **CVP Operations Console Server** - web-based tool to administer, manage, and configure CVP components.
   D. **CVP Reporting Server** – Informix database storing Vxml Server self-service data along with basic call events from CVP Call Server.

3. The following additional components are not part of the **CVP product** but may be required for a complete **CVP solution**. These are discussed in more detail on the following pages.
   A. **Cisco Gateways:**
      - **Ingress Voice Gateway** converts TDM phone signals to IP
      - **VoiceXML Gateway** interprets VXML documents, retrieves media files using HTTP, interacts with ASR and TTS servers using MRCP
      - **Egress Gateway** extends calls to the PSTN or a TDM ACD

   B. **Cisco Unified Communications Manager (UCM)** is a PBX to IP agent phones

   C. **Cisco Unified Contact Center Enterprise (UCCE** or **ICM)** makes routing and call treatment decisions

   D. **SIP Proxy Server (CUSP** - For SIP endpoints) allows for a centralized dialing plan, load balancing, and failover for SIP endpoints.
      -OR-
   E. **Cisco Gatekeeper** (For H323 endpoints) converts transfer labels to IP addresses. This allows for a centralized dialing plan, load balancing, and failover for H323 endpoints.

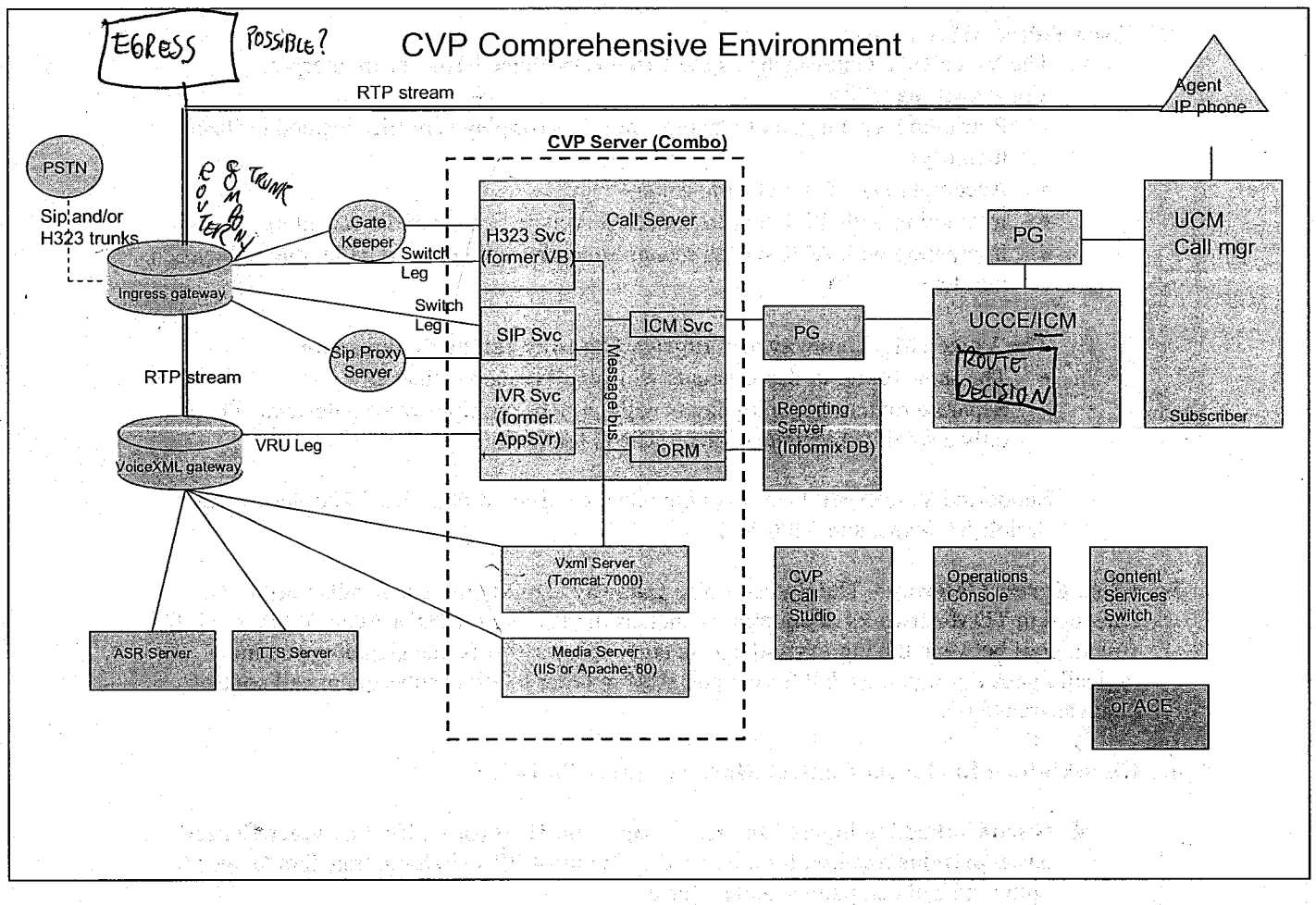
   F. **DNS Server** provides host name to IP lookup (and round robin load balancing)

   G. **Application Content Engine (ACE)** provides intelligent load balancing and failover for VXML Servers, ASR servers, TTS servers, Media Servers.
      - OR -
   H. **Content Services Switch (CSS)** provides intelligent load balancing and failover.

   I. **Third-Party Media Server** is a web server (IIS or Tomcat) to serve up static audio and grammar files

   J. **Third-Party ASR and TTS Servers** provides speech recognition and text-to-speech

   K. **Cisco Unified Information Center (CUIC)** is a centralized reporting tools for all CVP components. Especially UCCE/ICM and VXML Server Reporting Server.




1. **Ingress Voice Gateway**
   A. This is the point at which an incoming call enters the CVP system. It terminates TDM calls on one side and implements VoIP on the other side.

   B. Ingress Voice Gateways can convert TDM signals into **SIP** (or H.323 for upgrades only). Supported gateways include the Cisco 2800 Series, 3700 Series, 3800 Series, 5350XM, 5400 HPX, and 5400 XM. The Cisco AS5850 ERSC is also supported as an ingress voice gateway.

   C. **<u>CUBE for incoming SIP Trunks</u>**
   The use of third party SIP trunks with CVP is supported in CVP 8 with **Cisco Unified Border Element (CUBE)** which **runs as a feature** on a Cisco IOS Router.
   - Runs as a feature on Cisco IOS Router
   - Interconnects 2 different IP networks
   - Secures the enterprise edge – performs NAT for address hiding

- Performs transcoding between different codecs (G.711 and G.729)

2. **Cisco VoiceXML Gateway**
   A. The VoiceXML Gateway hosts the **Cisco IOS Voice Browser** to interpret VoiceXML pages by
      - Retrieving, caching, and playing .wav files (or play wav files located in flash memory).
      - Accepting DTMF input from callers
      - Interacting with TTS text-to-speech servers to deliver audio to callers
      - Interacting with ASR speech recognition servers to convert caller entered speech into text

   B. **The VoiceXML Gateway and Ingress Gateway can be deployed on:**
      - The same router in deployments with small branch offices
      - Separate routers in deployments with large or multiple voice gateways, where only a small percentage of the traffic is for CVP.

   C. Supported VoiceXML Gateways include the Cisco 2800 Series, 3700 Series, 3800 Series, 5350XM, and 5400 XM.

3. **Cisco Egress Gateway** – The Egress Voice Gateway is used only when calls need to be extended to TDM networks or equipment such as the PSTN or a TDM ACD. While the RTP stream goes between the ingress and egress voice gateway ports, the signaling stream logically goes through the CVP Server and ICM in order to allow subsequent call control (such as transfers).

4. **Cisco Unified Intelligent Contact Management** (ICM/UCCE)

   A. Cisco Unified Intelligent Contact Management **ICM** (or **Unified Contact Center Enterprise**) is required for advanced call control (IP switching, transfers to agents, outbound call campaigns, and so forth).

   B. ICM provides call-center agent management, call routing scripts, and database access.

   C. DNIS-based Call Routing scripts are created using the ICM Script Editor.

5. **CVP Call Server** (or CVP Server) – The CVP Call Server runs on a Windows Server to provide communication between the Gateways and ICM and to provide communication between VXML Server and the Reporting Server.
   The Call Server runs the following services:

   A. **SIP service** – implements **SIP Back-to-Back User Agent (B2BUA)** to accept SIP invites from ingress voice gateways and typically direct those new calls to a VXML Gateway port.

- After completing call setup, the B2BUA acts as an intermediary for subsequent call control. While the SIP signaling is hair pinned through this service, this service does not touch the RTP traffic.

B. **ICM Service** - This service is responsible for **all communication between ICM and the CVP Server components**: SIP Service, IVR Service, and H.323 Service.

C. **H.323 Service** (Formerly the CVP Voice Browser) - interacts with the IVR Service to relay call arrival, release, and transfer call control between it and the other H.323 components. This service is needed only for deployments using H.323. The H.3223 service interacts with the ICM via the ICM Service.

D. **IVR Service** - This service **creates the VoiceXML pages** to implement the microapplications based on ICM *Run VRU Script* instructions. This service runs an instance of Tomcat on port 8080 to create the VoiceXML pages and to receive http requests from the VoiceXML Gateway.

- The IVR Service maintains the VRU leg (in ICM parlance) to the VoiceXML Gateway.

6. **Peripheral Gateway (PG)** The Peripheral Gateway provides communication between external devices and ICM and can be considered a 'translator' allowing ICM to communicate with third party ACDs, PBXs, and VRUs.

A. Each Call Server maintains a **GED-125 Service Control Interface** connection to an ICM PG. GED-125 protocol uses a single socket connection to control many telephone calls.

B. Unified Communications Manager (UCM) also maintains a connection to ICM through a PG.

C. From the point of view of ICM, the Call Server is a voice response unit (VRU) connected to ICM, just as all other GED-125 VRUs are connected. **CVP is simply another routing client or VRU peripheral to ICM.**

7. **Cisco Unified Communications Manager (Unified CM)**

A. **Unified CM** is the main call processing component for managing and switching VoIP calls among IP phones.

B. Unified CM combines with ICM to form Cisco Unified Contact Center Enterprise (UCCE).

C. CVP interacts with Unified CM primarily to send PSTN-originated calls to Unified CCE agents. SIP gateway calls are routed to an available Unified CM SIP trunk. Similarly for H323 calls.

D. Unified CM is an optional component. Pure TDM-based call centers using ACDs, and CVP Standalone deployment models typically do not use Unified CM.

E. Unified CVP with SIP requires Unified CM 5.0 or later. Unified CVP with H.323 uses Unified CM 4.*x* or later.

8. **SIP Proxy Server (CUPS or CUSP) or Cisco Gatekeeper**

**SIP Proxy Server (SIP)**

A. The **SIP Proxy Server** is an **optional** component that routes individual SIP messages among SIP endpoints. It plays a key role in high-availability architecture for load balancing and failover among SIP endpoints.

B. The Cisco Unified Presence Server (CUPS Server), which has a built-in SIP Proxy function, and the Cisco Unified SIP Proxy Server (CUSP Server), which runs on an ISR gateway, are tested and supported with Unified CVP.

C. CVP may be deployed without a SIP Proxy Server, as some of the same functions can be provided by the CVP Server's SIP Service. If a SIP Proxy Server is not used, then Ingress Gateways and Unified CMs must point directly to Unified CVP.

**Cisco Gatekeeper (H323)**

A. The gatekeeper is an optional network element used by H.323 gateways for call routing. Most H.323 installations incorporate gatekeeper for dial plan configuration and bandwidth management.

B. Gatekeeper may be used to:

- Map specific dialed numbers to specific CVP Servers or VoiceXML gateways.
- Load-balance new calls to a set of CVP Servers or VoiceXML gateways.
- Route the transfer of callers from a VoiceXML gateway port to a Cisco IP Phone.

9. **DNS Server** – This component resolves hostnames to IP addresses anywhere in the network. It is often configured with multiple IP addresses for the same hostname for round-robin load balancing.

10. **Content Services Switch (CSS)**

A. The CSS is used for high-availability to provide intelligent load balancing and failover of VoiceXML Servers, Media Servers, and ASR/TTS Servers available to one or more VoiceXML Gateways.

B. CSS functionality is being replace by ACE.

11. **ACE: Application Content Engine**

A. **Application Content Engine (ACE)** is an alternative to the Content Services Switch (CSS) for server load balancing and failover with CVP 8.

B. ACE provides load-balancing services for HTTP, MRCP, and RTSP traffic, but not for call control signaling SIP or H.323 messages. It is used primarily to direct initial session requests for a particular type of service. There are four types of services:

- HTTP – Media Server (on TCP/IP port 80)
- ASR/TTS (on TCP/IP port 554)

- Unified CVP Call Server (http probe to the IVR service on TCP/IP port 8000)
- Unified CVP VXML Server (configure ACE for 'sticky' load-balancing), (port 7000)

12. **Third-Party Media Server**
    A. The Media Server component is a Microsoft IIS or Apache web server that serves prerecorded audio files, static VoiceXML documents, or external ASR grammars through HTTP request/response mechanism.

    B. Media Servers may be deployed simplex, as a redundant pair, or with a load balancer in a farm.

    C. The Media Server can be installed co-resident with the CVP Call Server and/or VXML Server.

13. **Third-Party Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) Servers**
    A. ASR and TTS Servers provide speech services to the VoiceXML Gateway using Media Resource Control Protocol (**MRCP**).

    B. Cisco does not sell, OEM, or support any ASR/TTS software or servers. Cisco tests CVP with **ScanSoft, Nuance, Loquendo,** and **IBM** offerings.

    C. Without CSS or ACE, a VoiceXML Gateway can support a maximum of two ASR/TTS Servers.

14. **Unified CVP Call Studio**
    A. **Call Studio** is the graphical script editor to create VXML Server applications. It is built on the open source Eclipse framework, and is essentially an offline Tool.

    B. Call Studio can execute on Windows 7, Vista, and XP workstations or servers.

    C. Because the Call Studio license is associated with the MAC address of the machine on which it is running, customers typically designate one or more data center servers for that purpose.

15. **Unified CVP VXML Server**
    A. Executes Call Studio applications by creating and sending VXML to the VoiceXML Gateway. Runs on a J2EE Tomcat or WebSphere application server. Integrates with J2EE components for back-end service integration.

    B. Can co-reside with a Call Server and/or Media Server (Windows 2003, 2008 Server ).

16. **Unified CVP Operations Console Server**
    A. The Operations Console Server is a Windows 2003 server that provides an Operations Console for the **browser-based administration and configuration** for

all Unified CVP product components, and it offers shortcuts into the administration and configuration interfaces of other Unified CVP solution components.

B. The Operations Console is **required** to:
   - Configure Call Server timers, services (H323, SIP, IVR, ICM), and parameters
   - Associate each VXML Server with a Call Server (for reporting purposes) and specify the data to send from the VXML Server to the Reporting Server,
   - Schedule backups and data retention times for the **Reporting Server**.
   - Configure Courtesy Callback

C. Optionally, the Operations Console can also be used to deploy files, scripts, licensing to different servers and gateways.

**17. Unified CVP Reporting Server**

A. The CVP Reporting Server is a Windows 2003 server that hosts an **IBM Informix** Dynamic Server (IDS) database management system to **provide historical reporting for a distributed self-service deployment.**

B. The database schema is fully published in the ***CVP Reporting Guide***.

C. The Reporting Server depends on the CVP Call Server to receive call records and reporting data from the IVR Service, the SIP Service (if used), and VXML Server.

D. The Reporting Server must be on the same LAN (not WAN) as the Call Server(s) and VXML Server(s) that it is servicing.

E. The Unified CVP *Operations Console* is used to implement Reporting Server administration and maintenance tasks, such as backups and purging. And to configure the VXML Server data to send to the Reporting Server.

F. CVP 8 includes the **Cisco Unified Intelligence Center (CUIC)** as a reporting engine for generating and displaying predefined reports from the Reporting Server and ICM databases. CUIC includes an Informix database that stores report definitions and generated report output. When CUIC is fully licensed it allows you to create custom reports and custom dashboards.

18. **Helix Server** (Optional) –You can use a Helix Server to create a broadcast stream of 8-bit uLaw wav files that can be played using one ICM PlayMedia node or one Studio URI to provide longer streams of music than separate 60 seconds wav files.

**ICM Configuration Manager: Create Network VRU Script**

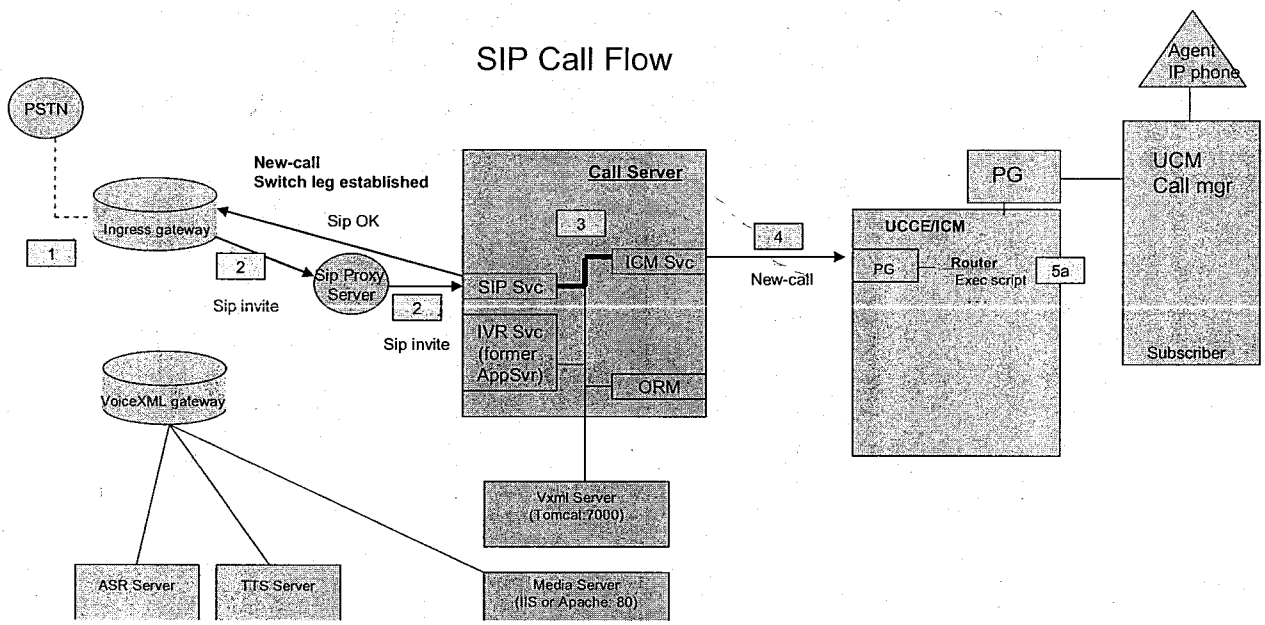
Name: PM,-1,A,5 (ie, audio file name comes from PV1)
Timeout: 99999999 (set to a duration longer than your audio stream)
Parameters: Y,0 (stream audio for this long. eg, 0=forever, 60=60s)

**ICM Routing Script**

- **SetVariable node** media_server <- rtsp://helixIP:554/streamname
- **SetVariable node** PV1 <- SoftMusic.mn
- **Run Ext Script node** (PM,-1,A,5) - Send error path of this node to another helix server by redefining **media_server** value.CSS and ACE don't support RTSP, so this mimics a backup

## Call Flow for a CVP Comprehensive Deployment Model

The Comprehensive deployment model provides routing and transferring calls across a VoIP network, IVR services, and queuing calls before being routed to a selected agent for a pure IP-based contact center.




### Initial Call Treatment and Self-Service

1. A call arrives at the **Ingress Gateway** which assigns a unique 36-digit Hexidecimal **Global Unique Identifier (GUID)** to the call. The ingress gateway must find a Call Server running the SIP service to which it can send the SIP invite.

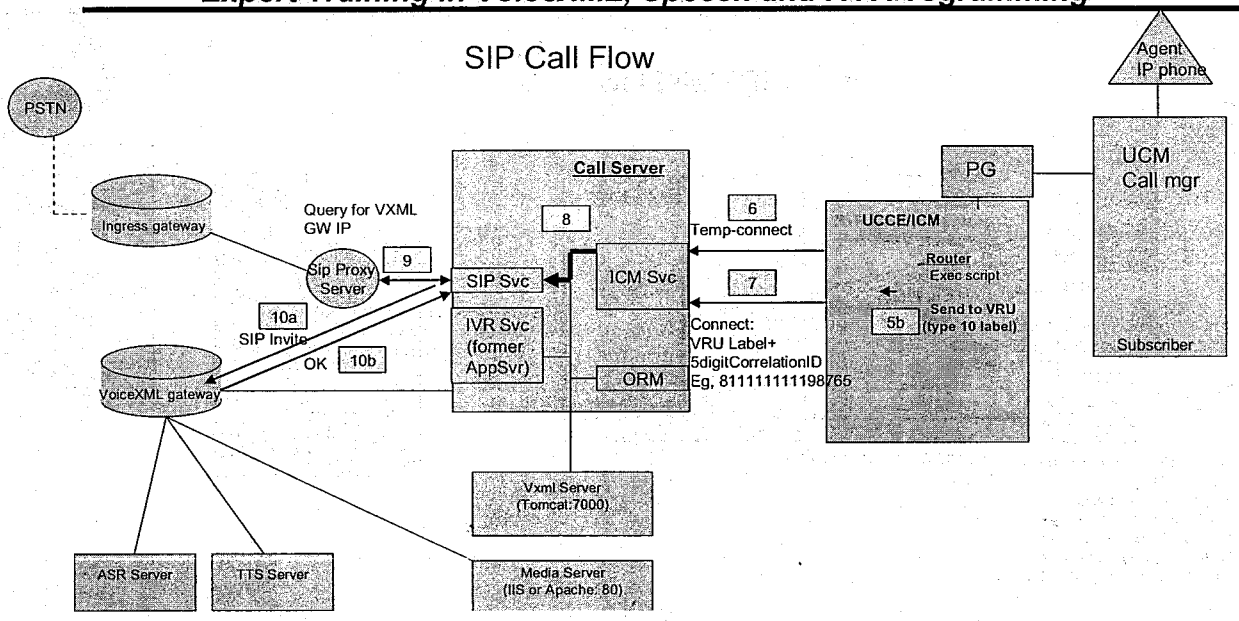
2. Finding an appropriate Call Server may be done (a) by configuring a dial peer on the ingress gateway to connect directly to the SIP service on a Call Server; or (b) by interrogating a SIP Proxy Server for the IP address of a Call Server running the SIP service.

   The **ingress gateway** sends a SIP invite to the **Call Server SIP Service**. This creates the **Switch Leg** of the call

3. The **Call Server SIP Service sends a new call request to** the Call Server ICM Service.

4. The ICM Service sends the new call request to ICM's VRU PG.

5. (a) This request causes ICM to run a routing script based upon the dialed number and other criteria.




**(5b)To provide self-service**: The ICM routing script utilizes a **Send to VRU** node to return a label to the VRU Routing Client to have the call sent to a **VoiceXML gateway**.

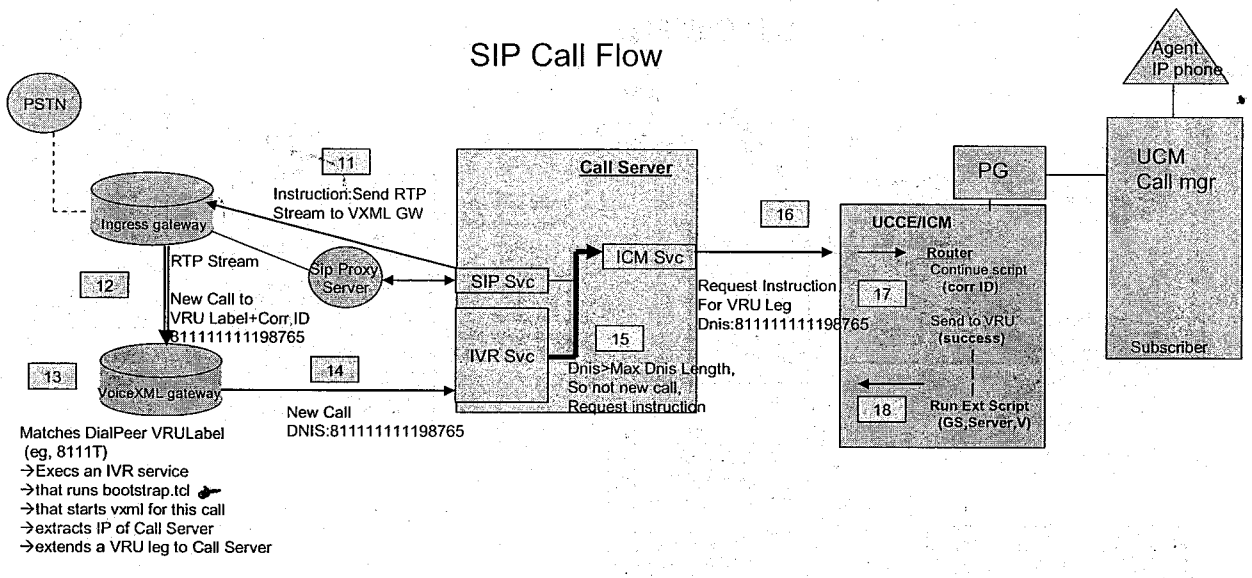

6. ICM returns a Temp-connect message to the Call Server

7. ICM then immediately returns a **Connect** instruction to the Call Server along with the VRU Label (Assume the VRU label is 10 digits: 8111111111) with a unique 5-digit (or less) correlation ID (example, 98767).

8. The ICM Service sends this connect instruction with the 15-digit label+CorrelationID (811111111198767) to the SIP service.

9. If the Ingress Gateways and VoiceXML Gateways are not co-located, then the **SIP Service** sends a request to the **SIP Proxy Server** to find the IP address of a VoiceXML gateway associated with the label returned by ICM. If using a co-located Ingress Gateway-VoiceXML Gateway, the SIP Service may be configured through the Operations Console) to return the IP address of the 'Originator' Gateway.

10. (a) The SIP Service sends a SIP invite to the VoiceXML Gateway; and (b) waits for the OK response.




11. The Call Server's **SIP Service** sends an instruction to the Ingress Gateway to direct the RTP audio stream to the VoiceXML Gateway.

12. The Ingress Gateway sends the audio as a **New Call** to the VoiceXML Gateway using the VRU Label+Correlation ID (returned from ICM) as the Dialed Number.

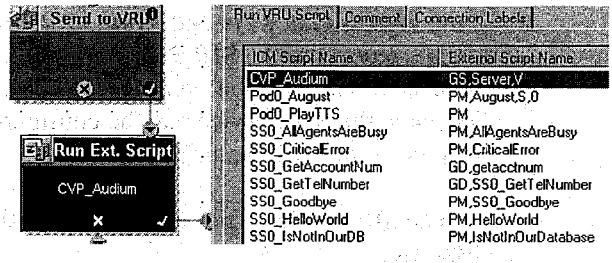
13. The VoiceXML gateway must have a dial peer configured to match the VRU Label (plus a wild card to allow for the extra digits of the correlation ID). This dial peer executes the bootstrap.tcl script that extracts the Call Server IP and starts the VXML Interpreter for this call.

14. The VoiceXML Gateway makes a New-Call request using VRU Label+Correlation ID as the Dialed Number. This connects with the **IVR Service** (formerly called the "App Server") of the Call Server to establish the **VRU Leg.**

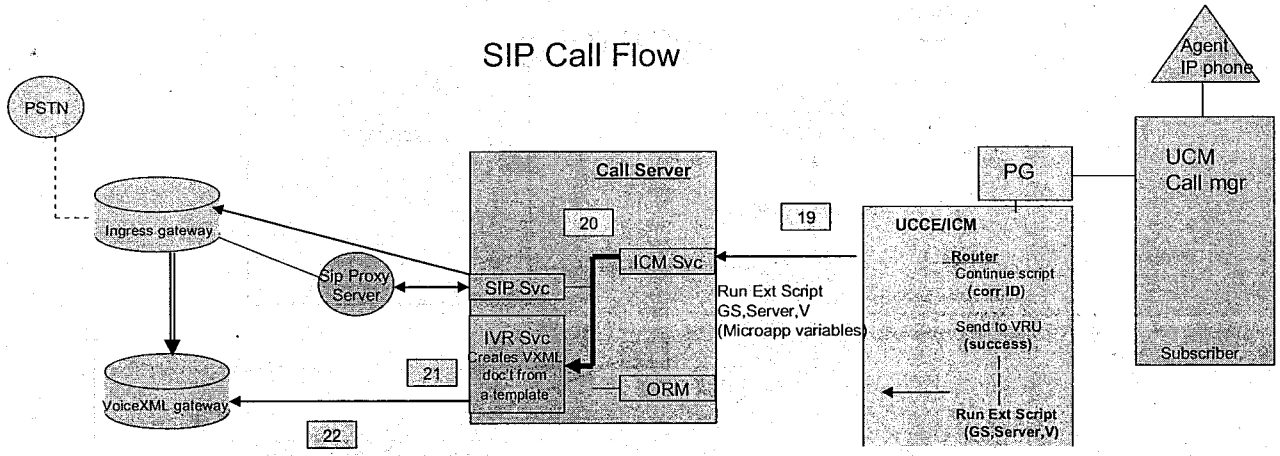
15. The IVR Service strips off the Correlation ID and converts this into a "request instruction" message for the call identified by the Correlation ID. This is sent to the ICM Service.

16. The ICM Service passes this along to ICM

17. ICM continues the routing script of the original call, continuing down the SUCCESS path of the Send to VRU node.
(This step has a 5s timeout)




18. The ICM routing script then executes a Run External Script node to execute a micro-application to provide VXML Self-Service. The "**GS,Server,V**" (GetSubdialog) script is required for Studio self-service. The name of the Studio application has been configured in an ECC variable.




**Sent to VxmlGateway:**

```
<vxml>
 <subdialog src="http://IP:7000/CVP/en-us/../Server?application=ABCbank&ani=xxx&dnis=YYY&callid=zzzzzzzz">
    <filled>
       <return namelist="caller_input FromExtVxml0, FromExtVxml1, FromExtVxml2,FromExtVxml3">
    </filled>
 </subdialog>
</vxml>
```

19. ICM returns the instruction to run the micro-application and the configured variables to the Call Server.

20. The Call Server's ICM Service passes this to the IVR Service.

21. All micro-applications cause the **IVR Service** to find a template of VXML code on the Call Server (C:\Cisco\CVP\conf\VXMLTemplates\) and insert the ICM data to configure that template. For the **GS,Server,V** the Call Server uses **GetSpeech-External.template**.

22. This VXML code is returned in the http response stream to the VoiceXML Gateway.

*Manually assigned in ToExtVXML*

**CVP8**

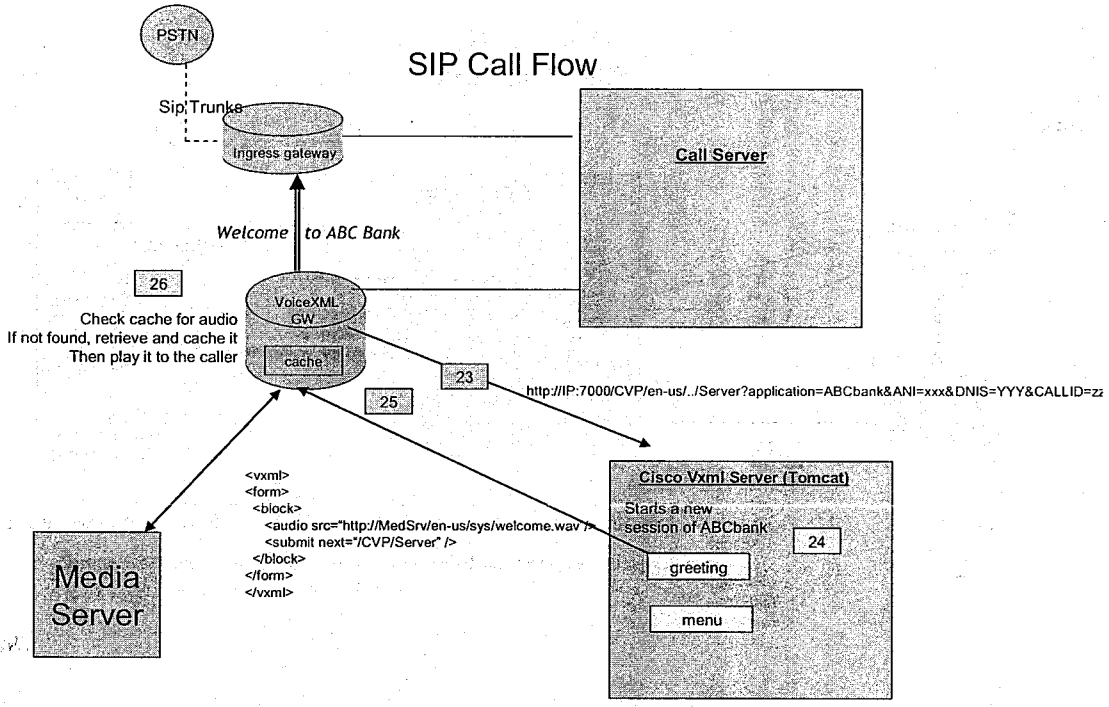
```
<vxml>
  <form>
    <subdialog   name="return" src=http://10.1.78.72:7000/CVP/en-us/../Server?application=HelloWorld&
                _dnis=555[illegible]34D56F789BA9871234">
      <filled>
        . . .
```

Automatically passed by CVP8

**CVP7**

```
<vxml>
  <form>
    <subdialog   name="return" src=http://10.1.78.72:7000/CVP/en-us/../Server?application=HelloWorld&
                dnis=5551212&ani=1212&callid= AE01234D56F789BA9871234">
      <filled>
        . . .
```

Manually assigned in ToExtVXML

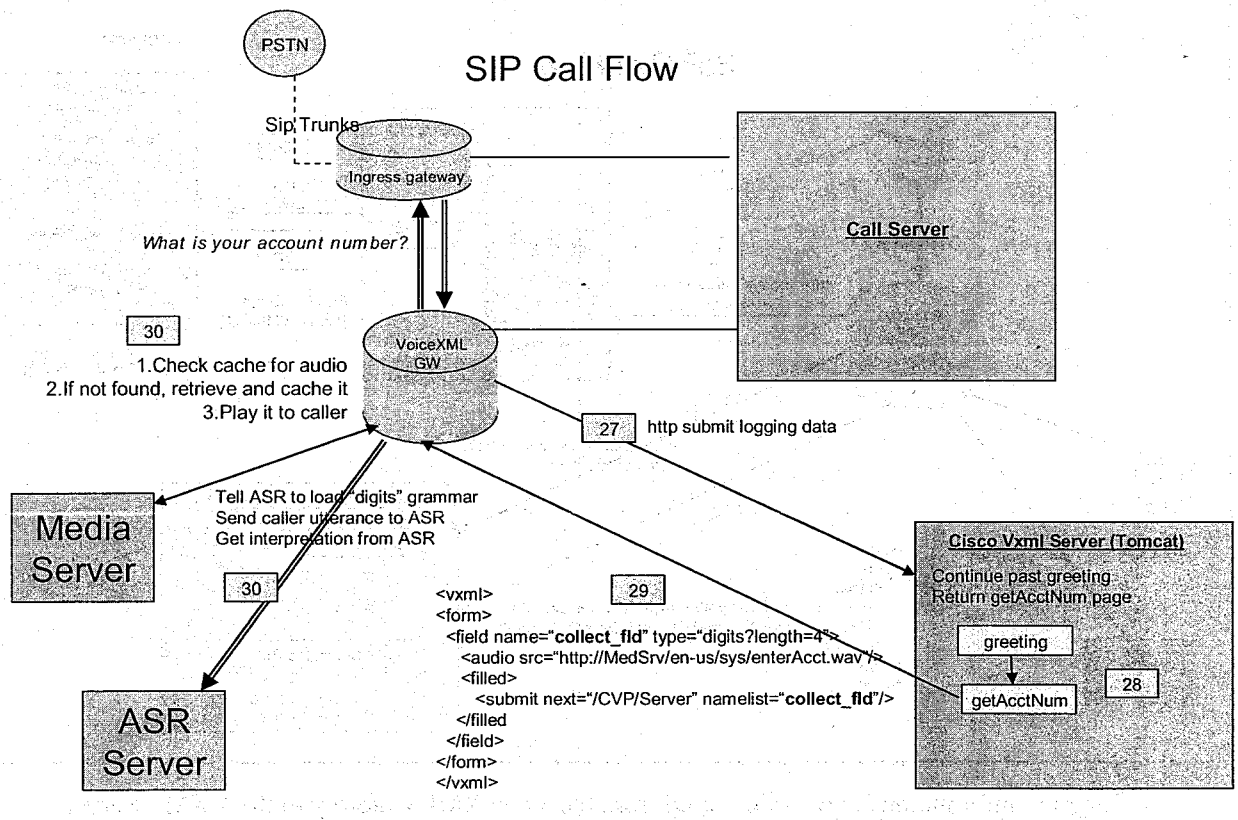



23. The VoiceXML Gateway pushes the current information onto a stack and executes the Subdialog statement to invoke the Vxml Server application.

24. The VoiceXML Server (Tomcat port 7000) starts a new session for this caller and begins executing the Studio application script.

25. Each interaction with the caller (audio broadcast or collecting caller input) sends a separate VXML page to the VoiceXML Gateway.

26. The VoiceXML Gateway tries to retrieve pre-recorded audio from its cache. If not found (or if expired) the VoiceXML Gateway retrieves and caches the audio and plays it to the caller.

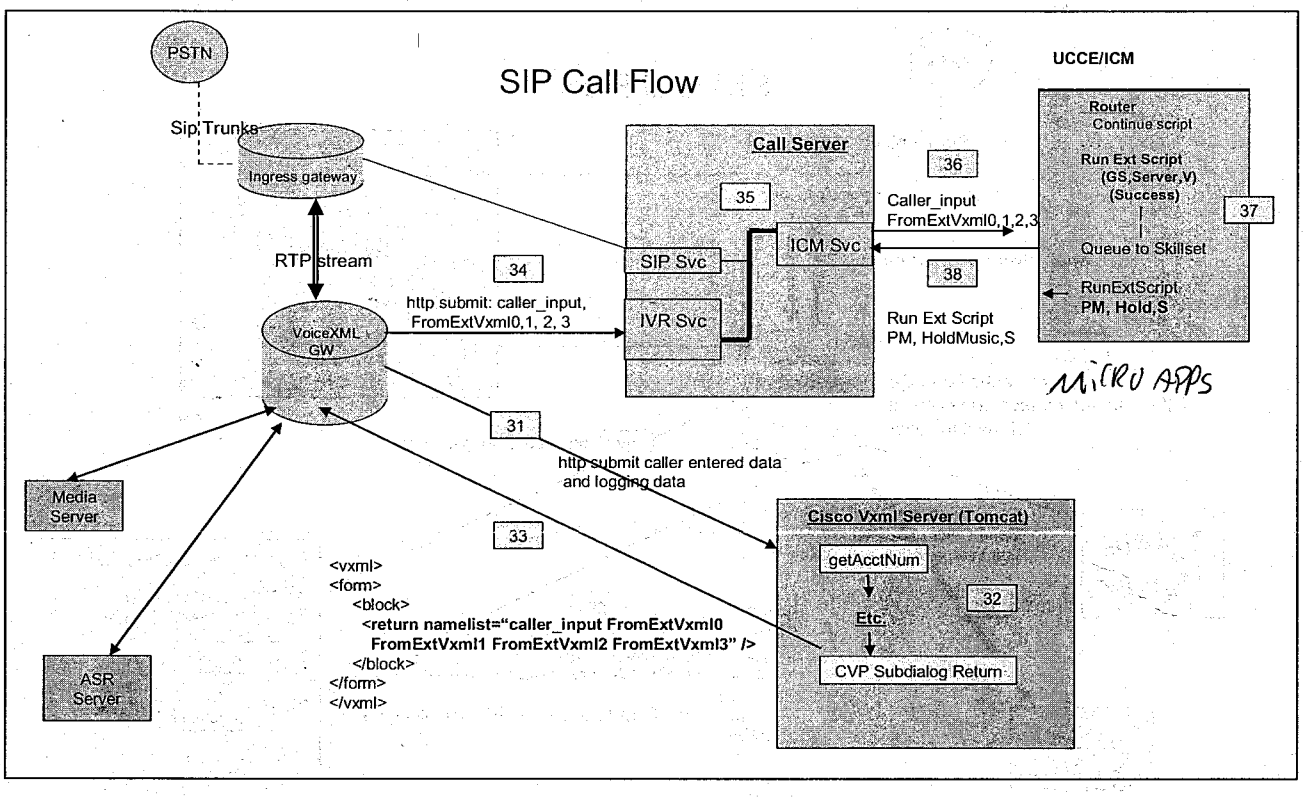



27. After each interaction with the caller, the VoiceXML Gateway submits data back to the Vxml Server for logging purposes and to continue the application's callflow.

28. The Vxml Server continues the application.

29. In this case, the application requests 4-digit input from the caller, using ASR or DTMF.

30. This is carried out by the VoiceXML Gateway. It uses MRCP to instruct the ASR server to load the Digits grammar and start listening. The VoiceXML gateway also checks the cache for the audio prompt and retrieves it from the Media Server, if necessary. The VoiceXML gateway plays the outgoing prompt and sends all audio input to the ASR server.




31. Logging information is always returned from the VoiceXML Gateway to the VXML Server for logging and to continue the application's callflow.

32. This continues until the Studio application executes a CVP Subdialog Return element. Five variables can be returned to ICM with data. **Caller_Input, FromExtVXML0, 1, 2, 3.**

33. The CVP Subdialog Return causes VXML Server to send a page to the gateway, returning from the existing 'Subdialog' with the data returned back.

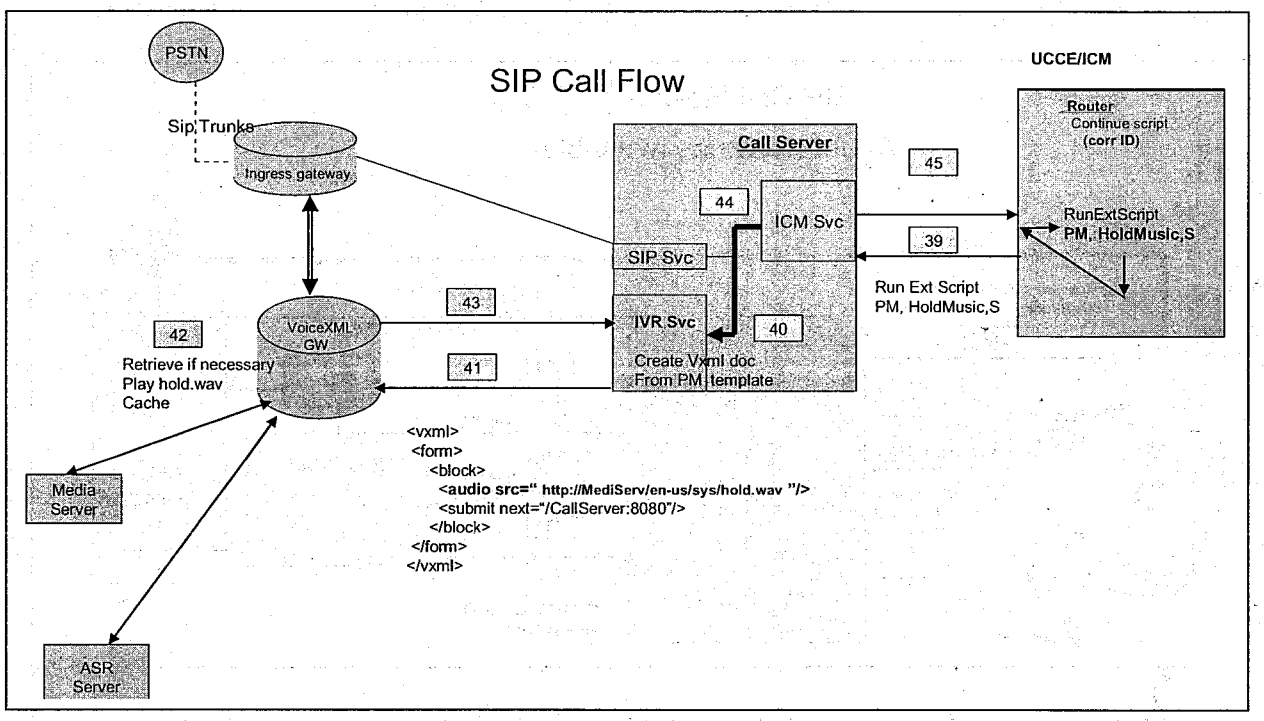
34. The gateway returns data from the VxmlServer to the Call Server's IVR Service.

35. The data and a *result=success* is sent from the Call Server's IVR Service, to the ICM Service

36. Data is relayed from the ICM Service to ICM into the caller_input variable and (optionally) into FromExtVXML[0], [1], [2], and [3] array. And the ICM routing script continues down the success path of the RunExtScript (GS,Server,V) node.

37. After analyzing the returned data, the routing script might change the call type and queue the call to a skill group.

38. While the caller is in a queue, ICM must provide music to the caller using another microapp such as the PlayMedia (PM, <audioFileName>). Note that you could send the call to another Studio application to play audio files until ICM interrupts and sends the caller to an agent.




39. This example shows ICM sending the request for the PlayMedia (PM, HoldMusic,S) microapp

40. The request is sent to the IVR Service which opens the PlayMedia template file (C:\Cisco\CVP\conf\VXMLTemplates\PlayMedia.template) and configures it based upon data received from the routing script.

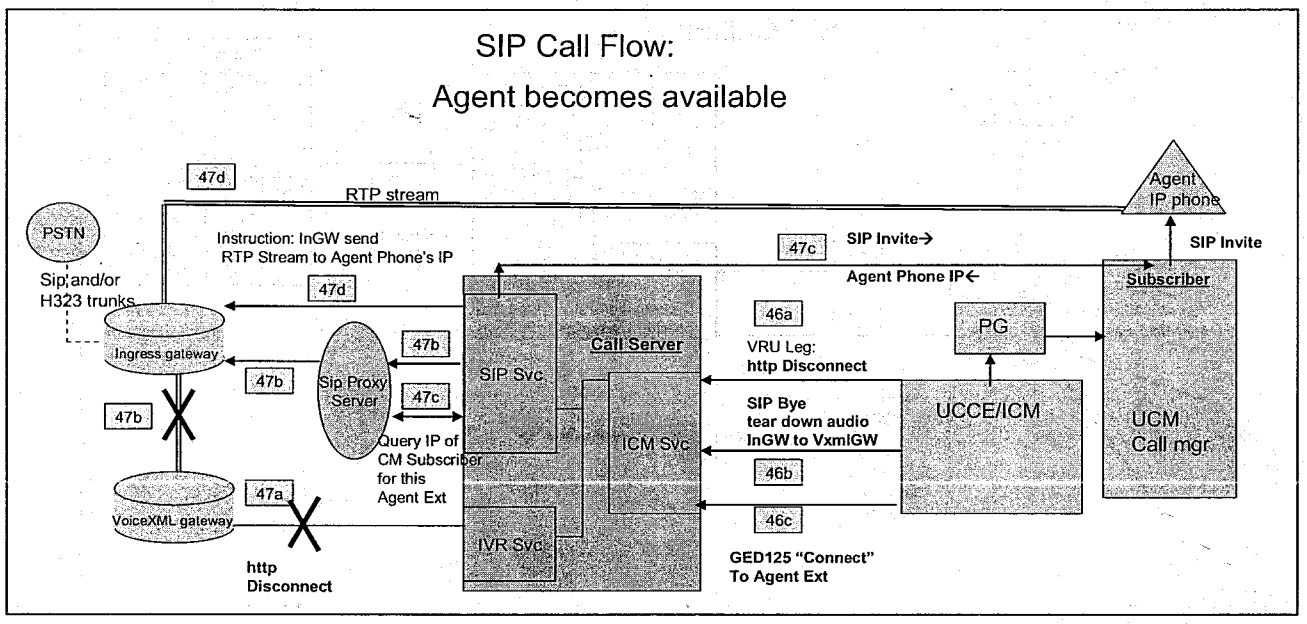
41. The VXML page is sent to the gateway to play the audio file.

42. The gateway plays the audio file from cache or retrieves it from the Media Server if necessary.

43. The VoiceXML gateway performs an HTTP submit to the Call Server's IVR Service with 'success'

44. This is forwarded to the ICM Service

45. This is sent to ICM. The routing script continues along the success path of the RunExtScript **(PM,HoldMusic,S)** node. The script should continue looping, playing one or more short audio files (60s) to the caller each time. Until an agent becomes available.

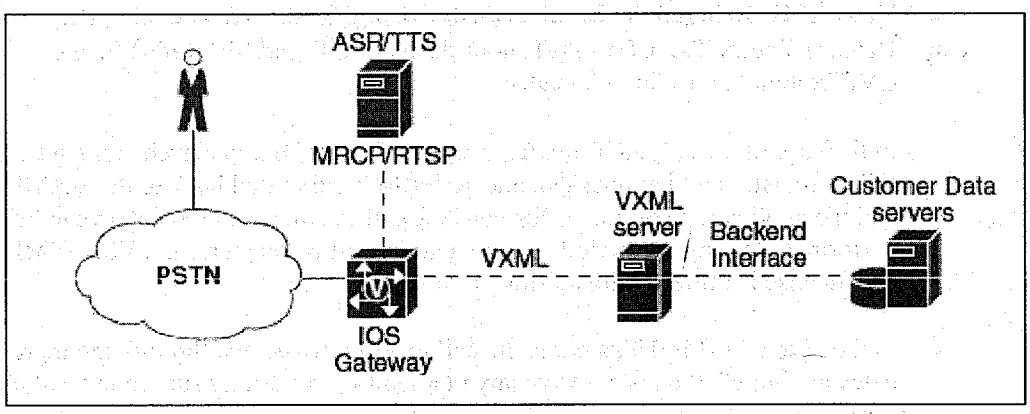



46. When an agent becomes available, ICM does the following:
    A. Sends an 'http disconnect' command to the IVR Service to tear down the VRU leg of the call.
    B. Sends a SIP 'BYE' to the SIP service to instruct the Ingress gateway to tear down the audio stream to the VoiceXML Gateway.
    C. Instructs the SIP service to send the call to the agent's extension

47. The instructions are carried out:
    A. The VRU leg of the call is torn down
    B. Sends a SIP 'BYE' is sent through the SIP Proxy Server to the Ingress Gateway which tears down the audio stream to the VoiceXML Gateway.
    C. The SIP service consults the SIP Proxy Server for the IP address of the Call Manager Subscriber to which this phone extension is registered.
        - The SIP service sends a SIP INVITE to that UCM Subscriber.
        - UCM forwards the SIP INVITE to the agent's IP phone.
        - Once the agent's IP phone returns a SIP OK, UCM sends the IP address of that agent phone to the Call Server's SIP Service.
        - The SIP Service sends a message to the Ingress Gateway to send the caller's RTP audio stream to this IP address.

## Standalone Deployment Model

The Standalone deployment model provides a standalone IVR solution for automated self-service. Callers can access CVP via local, long distance, or toll-free numbers terminating at CVP Ingress voice gateways that also contain the IOS voice browser. Callers can also access CVP from VoIP endpoints.




## Call Flow

1. A call arrives at the **ingress gateway** via TDM, SIP, or H.323. The gateway performs normal inbound POTS or VoIP **dial-peer matching**.

2. The selected VoiceXML gateway port invokes the **CVP self-service TCL script**.

3. The TCL script invokes **the CVP standalone bootstrap VoiceXML Document**, which performs an **HTTP request to the configured IP address of the VXML server**.
   http://IP:7000/CVP/Server?application=HelloWorld

4. The VXML Server runs the application specified in the HTTP URL and returns a dynamically generated VoiceXML document to the gateway.

5. The VoiceXML gateway parses and renders the VoiceXML document, playing wav files, invoking TTS and ASR servers, and performing DTMF detection.

6. As defined in the VoiceXML document, the **VoiceXML gateway submits an HTTP request containing the results of the caller input to the VXML Server**. The VXML Server continues running the application and returns a dynamically generated VoiceXML document to the VoiceXML gateway. The dialog continues by **repeating steps 5 and 6.**

7. The IVR dialogue ends when either the caller hangs up, the application releases, or the application initiates a transfer.

8. In addition to providing self-service, the Standalone VoiceXML deployment model can **transfer callers to another endpoint** – either VoIP (for example, Unified Communications Manager) or TDM (for example, egress voice gateway to PSTN or TDM ACD). **However, no IVR application data can be passed to the new endpoint with this deployment model**, therefore there will be no agent screen pop if the endpoint is a TDM ACD.

9. This model supports the following types of transfers:
   - **VoiceXML Bridged or Blind Transfer** (using Studio Transfer element)
   - **Release Trunk Transfer** (TNT, hookflash, TBCT, and SIP Refer) (using CVP_Subdialog_Return element)

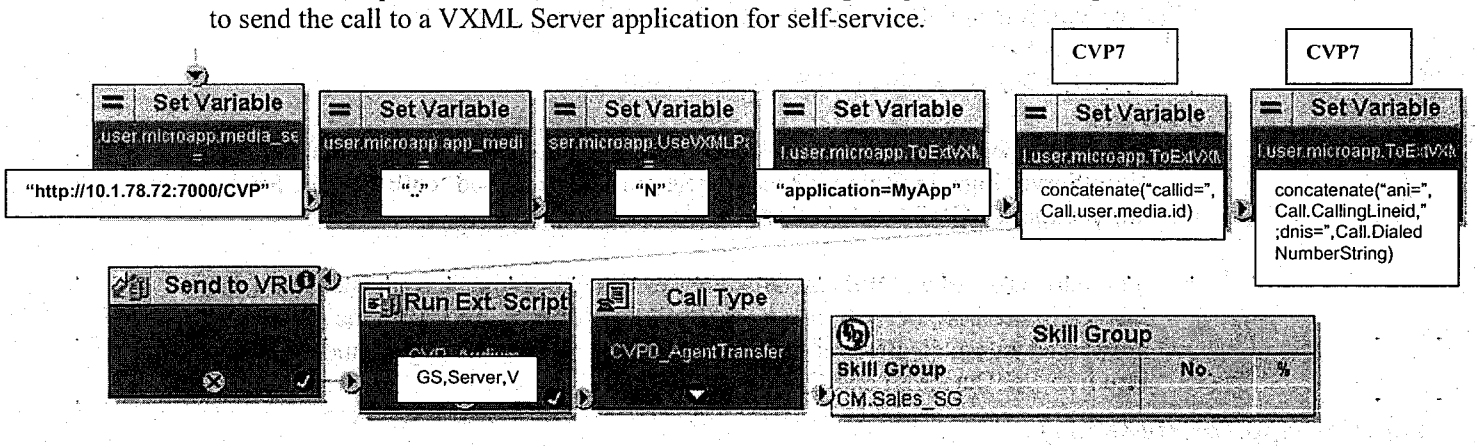
   - In the case of a **Bridged Transfer**, the outcome of the transferred call leg (transfer failed, transfer call leg released, and so forth) is submitted back to the VXML server and the application continues, further IVR call treatment and transfers can be performed. During the period of time that the call is transferred, a VoiceXML server port license is utilized with a bridged transfer.

   - In the case of a **Blind Transfer**, the call remains connected through the ingress voice gateway, but CVP does not have any method to provide any subsequent call control.

   - In the case of a **Release Trunk Transfer**, the ingress voice gateway port is released and no subsequent call control is possible.

## Configuring ICM Scripts to Invoke Vxml Server Applications

1. In a CVP Comprehensive environment, the ICM routing script must be configured as follows to send the call to a VXML Server application for self-service.




2. Use one **Set Variable** node to set each of the following **Call** variables:

A. **user.microapp.media_server** "http://***<vxml server IP address>***:7000/CVP"
- (Required) replace ***<vxml server IP address>*** with an IP address of host name of the **VXML Server**

B. **user.microapp.locale** **"en-us"**
- Usually been set to "en-us" in the Configuration Manager defaults or Ops Console. If so it's optional here.

C. **user.microapp.app_media_lib** **".."** (dot dot)
- (Required) This has usually been set to "app" by default in Configuration Manager, you MUST change its value.

D. **user.microapp.UseVXMLParams** **"N"** (Required) This specifies NOT to pass data using VXML Parameters, but to use HTTP Parameters instead.

E. **user.microapp.ToExtVXML[0]** **"application=*HelloWorld*"**
(Required) Replace ***HelloWorld*** with the name of the Studio application to execute. Case Sensitive!

**i) user.microapp.ToExtVXML[ 0], [1], [2], [3], [4] array**
- The **ToExtVXML** array allows you to pass data to the Studio application as **semi-colon** separated **name=value** pairs. Each ***name*** becomes the name of a Session variable in the Studio application with the assigned value.
- Example, ToExtVXML (Index 0) assign: "application=HelloWorld;language=spanish"

- The **maximum size** of the **ToExtVXML array is 5 rows x 210** characters.
- Check **Configuration Manager >List Tools >Expanded Call Variables** for the exact size on your system.

- You may use the ICM **concatenate** function to concatenate text and variables. This function takes up to 8 comma-separated arguments.
- Example: concatenate("RCK=";Call.RouterCallKey,";RCD=";Call.RouterCallDay)

Session

### ii) Passing the GUID, ANI, and DNIS

The **GUID (Global Unique Identifier)** is used to identify and track calls through the CVP system. It is initially assigned by the Ingress Gateway.

- The **GUID** is accessible in ICM as **user.media.id**.
- **(CVP 7)** You should pass the GUID to Studio as shown below to correlate ICM and VxmlServer logs. In Studio it becomes **Session Data** named '**callid**' in the Studio application. **In CVP 8 this is done automatically for you.**
- You may pass **ani** and **dnis** as shown below. They become Session Data named as labeled in the ToExtVXML array (eg, '**ani**' and '**dnis**'). **In CVP8** the ani and dnis are passed automatically, but they are stored into Session Data **_ani** and **_dnis** and also into CallData ANI and CallData DNIS.

**user.microapp.ToExtVXML[1]**
**concatenate("callid=", Call.user.media.id, ";ani=", Call.CallingLineID, ";dnis=", Call.DialedNumberString)**

3. **Send to VRU** – Use a Send to VRU node to send the caller's RTP audio stream from the Ingress Gateway to a VoiceXML Gateway. The switch leg between the Ingress Gateway and the Call Server (SIP Service) remains intact for telephony control commands.
   - When the call hits the VoiceXML Gateway, a **VRU Leg** is created between the VoiceXML Gateway and the Call Server (IVR Service). At that point, a request instruction command is sent to ICM which continues down the Success path of this node.

4. Use a **Run External Script** node– Once the call is at a VoiceXML Gateway, use the Run External Script node to execute VRU services to the caller using MicroApps. There are many microapps but only the **GS,Server,V** script will send the call to the VXML Server.
   - All network VRU scripts are created beforehand through the Admin Workstation's **Configuration Manager > List Tools > Network VRU Script List**.
   - All scripts have a default **3 minute timeout**. You **must** modify the timeout for longer duration Studio applications.
   - The ICM Script will be suspended until the Vxml Server application is complete.
   - See the ***Cisco CVP Configuration and Admin Guide (CAG.pdf)*** for full details on Microapps.

5. **Run External Script → Success Path** – When the VXML Server application ends successfully, data will be available in the Call variables:
   - **User.microapp.caller_input** (max 210 characters)
   - **User.microapp.FromExtVXML[0], [1], [2], [3]** (max 4 rows, 210 characters each)

- Consult the Configuration Manager's Expanded Call Variable List to determine the **actual** size of these variables on your system.

6. Data within these variables can be in any agreed-upon format. For example, data returned this way **skill=sales;acct=1234;** can be parsed as follows:

(skill): PeripheralVariable1 before("=",Call.user.FromExtVXML[0])
(sales): PeripheralVariable2 before(";",after("=",Call.user.microapp. FromExtVXML[0]))
(acct): PeripheralVariable3 before("=",after(";",Call.user.microapp. FromExtVXML[0]))
(1234): PeripheralVariable4 before(";",after("=",after(";",Call.user.microapp. FromExtVXML[0])))

7. **Run External Script → Failure Path** –ICM can interrogate the Call variable **user.microapp.error_code** for information about the cause of the failure. See from the ***CVP Configuration and Admin Guide*** reference manual for a list of the error codes.

8. After returning from the Vxml Server application, if the call is to be queued, then do the following:
   A. If using ASR in the VXML Server application, **you can attempt to manually free the ASR license** before queuing the call. Use a **Label** node with the checkbox **Target Requery** selected and a label that doesn't correlate to anything. Connect the Label node's error path to a **Send to VRU** node. Note that this may or may not work in your environment, consult Cisco TAC. In any case, the ASR license is definitely freed when the caller is transferred to an agent.
   B. Change the **Call Type** for service level and reporting purposes.
   C. Use some sort of queue node, such as **Queue to Skillgroup,** to queue the call to a Skillgroup
   D. To play music on hold using ICM Microapps, rather than VXMLServer, do the following:
      - Use a **Set Variable** node to assign to the **media_server** variable the actual Media Server URL:
        **user.microapp.media_server "http://mediaServerIP"**
      - Use a **RunExtScript** node to run a **PM** (Play Media) microapp to play music on hold.
        For example **"PM, Hold, S"** (**P**lay **M**edia, named **Hold.wav**, from the System prompts folder plays http://mediaServerIP/en-us/sys/Hold.wav)
      - Continue looping back to this element to continue playing music.
      - **NOTE** ICM has a (changeable) registry setting of a maximum queue time of 1 hour before dropping the call.

   E. When the agent is available, ICM will break out of this routing script and instruct the Call Server's SIP or H323 service to instruct the Ingress Gateway to tear down the path to the VxmlServer and direct the call to the Call Manager phone or PSTN.

**Comprehensive Configuration for Classroom**
**Ingress Gateway and VXML Gateway are Co-Resident**

**Comprehensive Configuration for Classroom**
**Ingress Gateway and VXML Gateway are Co-Resident**

```
version 12.4
!
no ip domain lookup
ip domain name yourdomain.com
ip host asr-en-us 10.1.78.31
ip host tts-en-us 10.1.78.51
ip host media 10.1.78.51
ip host asrtts-en-us 10.1.78.51
!
voice-card 0
 no dspfarm
voice service voip
 allow-connections h323 to h323
 allow-connections sip to sip
 sip
  bind control source-interface FastEthernet0/0
  bind media source-interface FastEthernet0/0
  rel1xx disable
  min-se  360
  header-passing
!
!voice translation-rule 1
 rule 1 /987654/ //  (Not needed for CVP8)
!
voice translation-profile block
 translate called 1
!
!Gateway has http client and cache for audio fetched from
!an http server (Media Server)

!Total size of cache for audio files
!(G711Ulaw - 1 Minute audio is 500 KB)
http client cache memory pool 15000 (KB)
!Only cache files if they are smaller than this
http client cache memory file 2000     (KB)

!audio in cache become 'stale' after this number secs
!unless overridden by VXML app or media server
http client cache refresh 3600          (s)

!error.badfetch if no connection made within this time
http client connection timeout 60       (s)
! error.badfetch if no response within this time
http client response timeout 60 (s)

!cisco no longer supports streamed audio
ivr prompt streamed none

!This is for taking recordings - either from callers; or to
update audio prompts; or for agent greeting
!One Minute is 500 KB
ivr record memory system <0-256000> (KB)
ivr record memory session <0-256000> (KB)
!
!ASR and TTS - MRCP URI's used to connect
ivr asr-server rtsp://asr-en-
us:4900/media/speechrecognizer
ivr tts-server rtsp://tts-en-us:554/synthesizer
!
mrcp client timeout connect 6
mrcp client timeout message 6
mrcp client session history duration 7200
mrcp client session history records 100
mrcp client rtpsetup enable
!
!Workarounds
!You need this for the beep to play when Recording
vxml version 2.0
! If you set vxml version 2.0, then this is required if you
want the VXML App to be notified of missing audio files
vxml audioerror
!
application
 service new-call flash:bootstrap.vxml
 paramspace english index 0
 paramspace english language en
 paramspace english location flash
 paramspace english prefix en
 !
 service cvp-survivability flash:survivability.tcl
 paramspace english index 0
 paramspace english language en
 paramspace english location flash
 paramspace english prefix en
 !
 service CVPSelfService flash:CVPSelfServiceBootstrap.vxml
 paramspace english language en
 paramspace english index 0
 paramspace english location flash:
 paramspace english prefix en
 !
 service bootstrap flash:bootstrap.tcl
 paramspace english index 0
 paramspace english language en
 paramspace english location flash
 paramspace english prefix en
 !
 service handoff flash:handoff.tcl
```

```
 service ringtone flash:ringtone.tcl
 paramspace english index 0
 paramspace english language en
 paramspace english location flash
 paramspace english prefix en
 !
 service cvperror flash:cvperror.tcl
 paramspace english index 0
 paramspace english language en
 paramspace english location flash
 paramspace english prefix en
 !
! Ingress Gateway -
! Dial Peers for 20xx go to Call Server at
! 10.1.78.10
 dial-peer voice 2070 voip
 destination-pattern 2070
 session protocol sipv2
 session target ipv4:10.1.78.10
 dtmf-relay rtp-nte
 codec g711ulaw
 no vad
!
dial-peer voice 2071 voip
 destination-pattern 2071
 session protocol sipv2
 session target ipv4:10.1.78.10
 dtmf-relay rtp-nte
 codec g711ulaw
 no vad
!
!
```

```
!VoiceXML Gateway -
!Label coming back from ICM
! 8111111111+correlationID match this dial peer that invokes the
!bootstrop.tcl to create the VRU Leg for this call

dial-peer voice 81 voip
 description Label from ICM
 service bootstrap
 incoming called-number 81T
 dtmf-relay rtp-nte h245-signal h245-alphanumeric
 codec g711ulaw
 no vad
!
dial-peer voice 987654 voip  (Not needed for CVP8)
 description Fixes Programming Workaround by Blocking 987654
 translation-profile incoming block
 incoming called-number 987654
!
dial-peer voice 9191 voip
 description SIP Ringtone Dial-Peer
 service ringtone
 incoming called-number 91919191
 dtmf-relay rtp-nte h245-signal h245-alphanumeric
 codec g711ulaw
 no vad
!
dial-peer voice 9292 voip
 description SIP Error Dial-Peer
 service cvperror
 voice-class sip rel1xx disable
 incoming called-number 92929292
 dtmf-relay rtp-nte h245-signal h245-alphanumeric
 codec g711ulaw
 no vad
!
!Agent Phones using ICM Label 40xx
dial-peer voice 4070 voip
 destination-pattern 4070
 session protocol sipv2
 session target ipv4:10.1.78.70
 dtmf-relay rtp-nte
 codec g711ulaw
 no vad
!
dial-peer voice 4071 voip
 destination-pattern 4071
 session protocol sipv2
 session target ipv4:10.1.78.71
 dtmf-relay rtp-nte
```

**Chapter 1 Review Questions**

1. Match each component description on the left to the component name on the right (use each item once).

| | |
|---|---|
| E 1.Runs routing scripts and makes routing decisions. | A. Ingress gateway |
| C 2.Interprets VXML documents | B. The Switch Leg |
| G 3.Converts labels into IP addresses for SIP endpoints | C. VoiceXML gateway |
| A 4.Converts telephone TDM signals into VoIP | D. VXML Server |
| D 5.Runs Call Studio applications to create VXML dynamically | E. ICM |
| I 6.The server that runs the SIP, ICM, and IVR services. | F. Call Manager |
| F 7.Acts as a PBX to agent IP telephones | G. SIP Proxy Server |
| K 8.Translates messages between ICM and ACDs/PBXs | H. Content Services Switch or ACE |
| H 9.Load balances among servers and provides fail over | I. CVP Call Server |
| B 9. The leg of the call between the ingress gateway and the call server for all telephony commands. | J. The VRU Leg |
| J 10. The leg of the call between the VoiceXML gateway and the call server for http interface to request and receive VXML documents associated wit MicroApps. | K. Peripheral Gateway |

2. **The ICM Routing Script:**

Answer the following to send the call to a VXML Server application named "**Bank**" running on a node named **VxmlSvrHost**, port **7000**.

A. What do you assign into the **user.microapp.media_server** variable?
"HTTP://VXMLSRVHOST:7000/CVP"
B. What do you assign into the **user.microapp.locale** variable?
"EN-US"
C. What do you assign into the **user.microapp.app_media_lib** variable?
""
D. What do you assign into the **user.microapp.UseVXMLParams** variable?
"N"

(continued)

E. What do you assign into the **user.microapp.ToExtVXML[0],[1],[2]** variables? Can you combine these or must you assign the data into array entries 0,1, and 2?

"APPLICATION = BANK ; LANG = SPANISH"

F. Which node sends the call from the Ingress gateway to the VoiceXML gateway?

SEND TO VRU

G. Which node instructs the VoiceXML gateway to run the VXML Server application? Which microapp must you select?

RUN EXT SCRIPT (ES, SERVER, V)

H. If the routing script continues down the success path, which variables should you check for the information and data returned from the self-service? What format is it in?

FORMEXTVXML [0], [1], [2], [3] AND CALLER_INPUT

I. If the routing script continues down the failure path, which variables should you check for the information about why it failed?

USER. MICROAPP. ERROR_CODE

J. After returning from VXMLServer self-service, what 2 steps are required in the ICM routing script to play music on hold while the caller is queued?

SET VARIABLE NODE: SET THE MEDIA_SERVER VARIABLE

RUN EXTERNAL SCRIPT NODE, (PM, HOLD)

[End of Chapter 1]

# Chapter 2

# Call Studio: Writing and Testing a Studio Application

- Introduction to Call Studio
- Creating a Project
- Using CVP Subdialog Start and Return Elements
- Using an Audio Element to Speak to Callers
- Deployment and Testing
- Understanding Cisco Audio Caching

## CVP Call Studio

**CVP Call Studio** is a development platform built on **Eclipse** for the creation of voice applications. CVP Call Studio provides a drag-and-drop graphical user interface (GUI) for the rapid creation of advanced voice applications.

## Working with Call Studio

To create a new Call Studio voice application (or project):

- **Right-click within the Navigator** pane and select **New > Cisco Call Studio Project** or select from the top-level menus **File > New**.
- Enter a project name (example, **MyApp**), click **FINISH**.
- Note that project names should start with an alphabetical character, contain alphanumeric characters and underscores only, is case-sensitive, and should not contain blank spaces or dots.
- The new application/project will appear in the Navigator window.

## Call Studio Application Source Code Directory

1. By default, Call Studio stores applications on the hard drive into:
   **C:Cisco/CallStudio/eclipse/workspace**

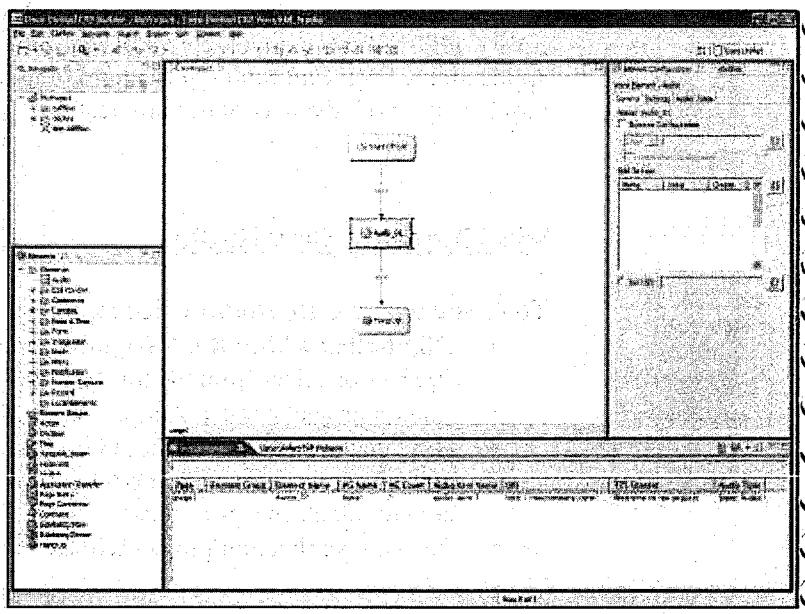
2. To change the default workspace for Studio, modify the **–d** parameter in the file **C:Cisco/CallStudio/startStudio.cmd**

3. The application consists of a number of files and folders that define the configuration and layout of the Studio project. The application's folder can be viewed in the Navigator pane of Call Studio.

4. In the **Navigator view,** right-click the project name to
   - Rename
   - Copy
   - Paste
   - Delete
   - Import an existing Call Studio project

5. To open an application in the Workspace (Callflow Editor) double-click the application's **app.callflow** in the Navigator pane.

## Call Studio "Views" or "Panes"



CVP Call Studio is separated into different panes or "views".

- **Navigator View** – displays projects and their subfolders. A CVP Studio project is stored as a number of folders and files that together define the application.

- **Elements View** – the element palette for creating the application's call flow. The main types of elements in Call Studio: **Action, Decision, and Voice**. These are combined with other less used elements to represent a voice application.

- **Workspace (or Callflow Editor)** – Edit the callflow in this pane. To display the workspace where you build the application, you must double-click the project's **app.callflow** in the Navigator view.

- **Element Configuration View** – After dragging an element into the workspace, double click the element. The right side displays the Element Configuration view, where you can configure the element.

- **Outline View** – This shares the rightmost pane with the Element Configuration View. This clickable alphabetical display of pages and elements is **useful for navigating very large applications**.

- **Problems View**– When validating or deploying the project, error messages appear at the bottom of Call Studio in the Problems view.

- **Prompt Manager View** – View and edit prompts in this view.

**DEBUG TIP:** To return to this default set of windows (Builder Perspective), select the top row menu button **Window > Reset Perspective**

## Working with Elements

1. Add an element
   - Drag an element from the Elements view into the workspace.

2. Delete an element
   - To delete an element into the workspace, select the element in the workspace.
   - Either **Right-click/select Delete** or press the **Delete** key on your keyboard.

3. Connections Between Elements - "Exit States"

A. **Connect elements** by connecting their **Exit States** (the labeled connection lines)
   - Most elements have one exit state (done or next). But some have multiple exit states.
   - **All exit states MUST be connected before Studio will allow you to Deploy the application.**

**B. Connect Exit States**
   - **Right-click** the source element, click the **Exit States menu**
   - Select the exit state to connect (some elements may have more than one exit state).
   - Place the cursor on the destination element and **Click** (left-click)

**C. To delete a connection:**
   - Select the Exit State connection line
   - Right-click, choose Delete.

D. **To bend a connection**:
   - Select the Exit State connection line
   - Under the name of the Exit State, a tiny square appears, this is an 'elbow joint'
   - Drag the 'elbow joint' to create a bend in the Exit State
   - Each line segment can be bent again and again

E. **Redirect a connection:** it is easier to move a connection than to delete and create a new one. To redirect a connection,
   - Click the Exit State connection line so the arrowhead points to a tiny square
   - Place your cursor on the tiny square
   - Drag and drop the square and arrow head on a new element.

F. **All Exit State must be connected.** An element displays a warning sign ( **!** ) if an exit state is not connected.
   - Studio will not allow you to 'Deploy' ('Compile') or test the application unless all exit states are connected.

4. Naming Elements
   - Elements should be given **meaningful names** to make code readable.

- Names must begin with a letter, and may contain letters (case sensitive), numbers, underscores, and spaces.
- The Reporting Server allots **51 characters** for the element name field.
- Every element in the application must have a unique name.
- To name the element right-click on the element and select **Rename** (or click the element and wait 1 second).

5. Configuring an element
   A. Almost every element requires configuration and each element's configuration is different. To view the **Element Configuration** view, you may have to :

      - Double-click the element
      - The right-most pane displays the Element Configuration view

   B. Almost every Element Configuration view contains:

      - The **General** tab for adding information to the Activity Log
      - The **Settings** tab has settings that make this element unique. Settings marked with an asterisk * are required to have an entry. Those marked with '+' may be repeated.
      - The **Audio** tab allows you to configure audio prompts to speak with the caller
      - The **Data** tab allows you to create or assign values to variables.
      - The **Hotlinks** tab allows you to configure DTMF or utterances that throw an event or exit down a customized exit state. Example '***go back***'

## Documenting Your Code

### Commenting Elements

1. To add comments to an element, **Right-click** and select **Edit Comment**
2. Elements with comments display a **red triangle** ▲
3. To view the comment, Right-click and select **Edit Comment**.

### General Comments and Notes

- Near the bottom of the Elements view is a **Comment element**.
- Comments may be used for general notes and documentation and are not attached to an element.
- Drag in a Comment element and **Double-click** to open the **Edit Comment** editor.
- Comments appear on the page where they are created.

### Audio Comments

Each Audio prompt allows you to right-click and add comments. These comments display in the Documenter printout.

### Documenter

1. The Documenter allows you to print your application to a file in Rich Text Format (rtf).

2. To invoke the Documenter, **right-click the project name** in the Navigator view and select **Documenter**. Select the options to include and then specify the destination file location and file name.

3. Viewing the rtf file in **Microsoft Word** or **OpenOffice Swriter** (free, open source editor) allows you to view the callflow diagrams. Viewing with Notepad or WordPad does NOT display the diagrams.

4. Documenter output will include comments added anywhere in the Studio application, diagrams, element configuration, audio prompts, and global properties.

5. A sample Documenter printout is included near the end of this student guide.

## Working with Eclipse

1. Eclipse has most of the familiar keyboard shortcuts
   - Cut Ctrl+x
   - Copy Ctrl+c
   - Paste Ctrl+v
   - Save Ctrl+s
   - Undo Ctrl+z
   - Redo Ctrl+y
2. See the **Help/Key Assist...** menu for a full list of keyboard shortcuts.
3. To maximize a window pane, **Double-click** the tab at the top of the page. **Double-click** again to toggle back to the original size.
4. To reset the windows to the original view (perspective), select the top menu button **Window > Reset Perspective**.
5. For an online Studio reference manual, select **Help/Help Contents**
6. Eclipse (and hence Call Studio) supports CVS versioning systems for maintaining multiple versions of source code. It also supports Team based features such as storing code onto a Server and allowing developers to 'check out' applications or portions of applications making them 'read only' to others.
7. Eclipse is open source software supported by the Eclipse Foundation (www.eclipse.org).

## Writing Your First Application with Call Studio

Create a new Call Studio voice application (or project):

1. Select **File > New > Cisco Call Studio Project**
2. Enter a project name (example, **MyApp**), click **Next**.
3. You will now see **Four Project Properties** pages that may be configured now or later.
    a. **General Settings**, such as the Voice Gateway (voice browser and speech recognizer), the Session Inactivity timeout and the Loggers. **Click Next.**

    b. **Audio Settings**, such as the **Default Audio Path** on which to append audio file names; and default error messages. **Click Next.**

    c. **Root Document Settings**, such as default values for VoiceXML properties (e.g., interdigittimeout). **Click Next.**

    d. **Endpoint Settings** – custom code that executes at the start or end of the visit to this application. **Click Finish.**

## Project Properties

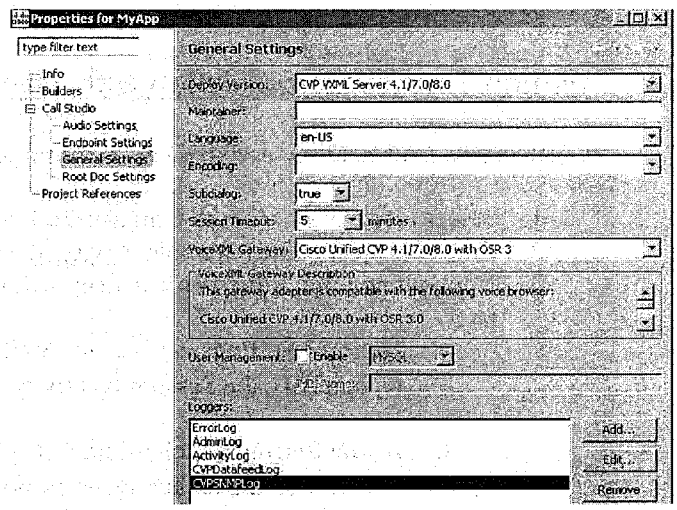
Modifying **Project /Properties**:

1. Properties (the 4 windows of Settings above) can be set when a project is first created.

2. After the project is created, properties are also accessible by **right-clicking** on the **project name** in the **Navigator View** and selecting **Properties** from the menu.

3. Properties are also available from the **Project** menu at the top of Studio, select **Properties**.

## General Settings




1. **Maintainer**. (Optional) Enter an e-mail address of the application administrator. If the voice browser encounters errors (missing audio files or other configuration problems) it sends a message to this e-mail address. You must then configure the VXML Gateway to send email.

2. **Language**. (Optional) Select **en-US** or select or enter the language for multi-language pre-recorded audio, ASR, and TTS. This displays in the VoiceXML document's **xml:lang** attribute. Default: **en-US**.

3. **Encoding**. (Optional) XML documents can contain non-ASCII characters, like Norwegian æ ø å , or French ê è é. To avoid errors, specify the XML encoding as one of the following single byte encoding (UTF-8 or ISO-8859-1) and double-byte encoding (UTF-16) for non-ASCII characters.

4. **Subdialog (true/false)**. Leave this set to **true** because CVP invokes the VXML Server application as a Subdialog.

5. **Session Timeout**. This value sets the length of time a session on the application server is allowed to linger if no activity from the VXML Gateway is detected. 5 minutes

6. **VoiceXML Gateway**. This sets the **voice browser and speech recognizer** the application will be executed upon.

7. **User Management**. This activates a lightweight User Management system which tracks calls based upon CallData.ANI and requires access to MS SQL Server or MySQL database. Tomcat must be configured to access the database using JNDI. VXML Server creates the database tables at runtime.

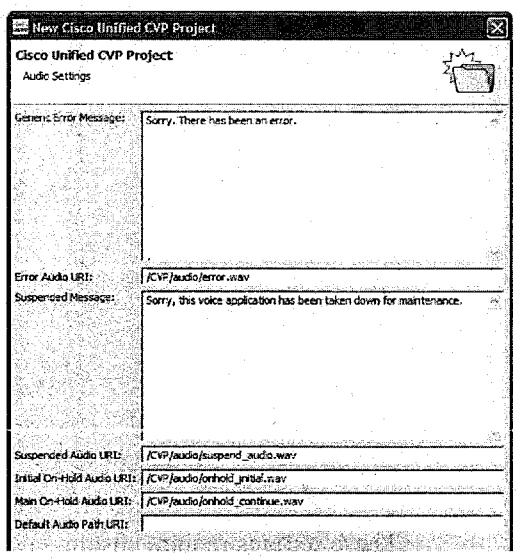
8. **Loggers**. Configure logging levels for the application. Refer to the ***VoiceXML Server User Guide for Cisco CVP*** for more information on specific loggers. **WARNING!** If you have no reporting server, you should remove the **CVPDatafeedLog** in order to deploy and run on the VXML Server without warnings.

## Audio Settings

1. This window contains the location of **default error messages** played to the caller. You may re-record the audio or point to recordings in a different location.

2. The audio files listed are on the VXML Server in the directory **C:\Cisco\CVP\VXMLServer\Tomcat\webapps\CVP\audio**

3. This window is also where you specify the **Default Audio Path URI** which is prepended to audio file names used in the application. This "default" may be changed during the call flow with the Application Modifier element.



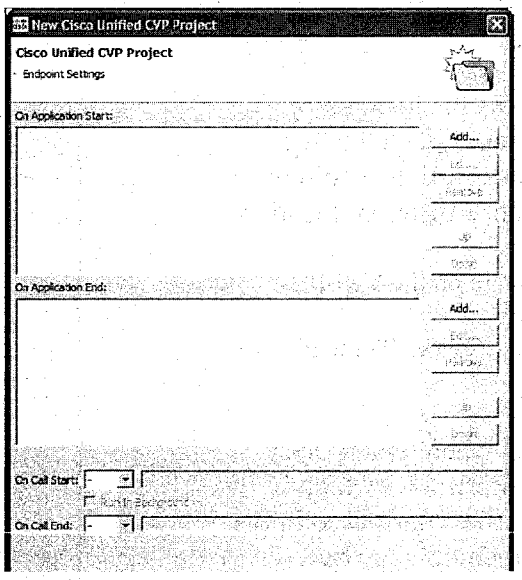

*A.* **Generic Error Message and Error Audio URI.** A pre-recorded audio file played when the VXML Server encounters an unexpected error during the course of a call. Default recording says: *'I'm sorry we're experiencing difficulty, please call back at a later time'*
- When the audio file is provided, the **Generic Error Message** will only be used as a TTS backup message when this audio file cannot be played.

*B.* **Suspended Message** and **Suspended Audio URI**. This is a URI to a pre-recorded audio file containing the message to play when the application is suspended. The TTS message above is played only when the audio file specified here is not found or corrupted. *'I'm sorry, the requested application is currently down for maintenance. Goodbye'*

C. **Initial On-Hold Audio URI / Main On-Hold Audio URI**. Each simultaneous call to VXML Server takes up a VXML Server port. The license purchased from Cisco specifies how many ports the system can support. If a call is received when all ports are taken, VoiceXML **Server queues the caller on hold**, playing the audio referenced in **Initial On-Hold Audio URI**. *'We are currently experiencing heavy call volume. Please hold, your call will be answered in the order in which it was received. (8 seconds silence)'*
- When done playing, VXML Server checks if a port is available. If not, it will play the **Main On-Hold Audio URI** followed by another check. *'Please continue to hold (8 seconds silence)'*
- The cycle continues by repeatedly playing the main on hold audio and checking for a port. Once a port is available, the caller starts interacting with the application.

D. **Default Audio Path URI**. Enter a URI to a path containing the audio files for this voice application. This path will be prepended to audio files used in the application that specify to use the Default Audio Path. (A trailing slash is allowed but not required.)
**Example:** **http://10.1.78.12/en-us/sys**
For IIS, this corresponds (by default) to: **C:/Inetpub/wwwroot/en-us/sys/**

## Endpoint Settings




These settings require custom java components for added functionality when the application is loaded/unloaded from memory (On Application Start/End) or when each caller Enters/Exits the application (On Call Start/End).

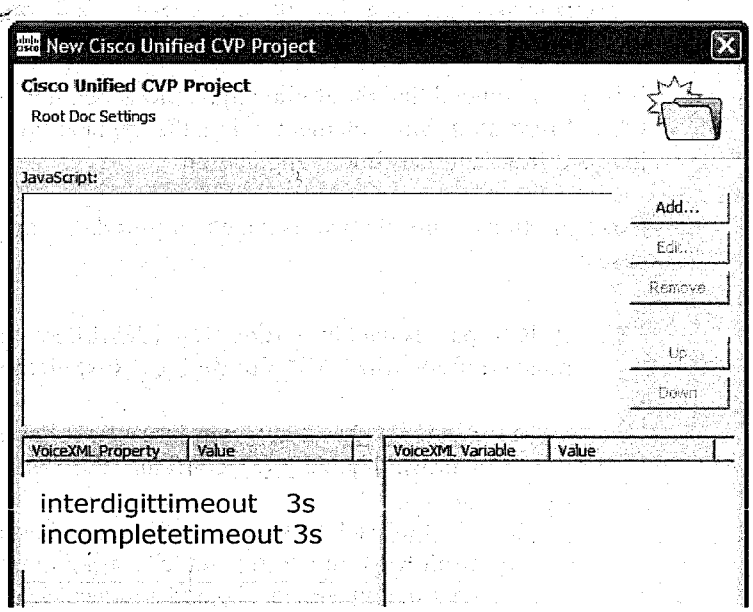
1. **On Application Start.** The On Application Start section specifies the Java classes that are executed when the voice application is started on the VXML server (boot time or deployApp or updateApp).
- Useful for creating Global (persistent) variables used by all calls.

2. **On Application End.** The On Application End section specifies the Java classes that are executed when the voice application is stopped on the VXML Server. (shut down or releaseApp or suspendApp).
- Useful for deleting Global (persistent) variables.

3. **On Call Start**. This specifies a Java class or URI to be accessed when the application is entered by a caller.
- Useful for creating Session variables based upon ICM-passed data, contents of a properties file, database retrieval.

4. **On Call End**. This specifies a Java class or URI to be accessed no matter how the application visit for a caller ends: caller hang up, return to ICM, an unrecoverable error, application transfer to another Studio application, etc.
- Useful for creating customer detail records or reporting based upon callflow activity.

## Root Document Settings




1. **JavaScript**. Specify custom JavaScript content that will be inserted into the application root document. All script content listed in the edit box is inserted within VoiceXML <script> tags by VoiceXML Server and therefore, <script> tags should not be defined in the script content.

   - **Note: JavaScript entered here requires custom Java voice elements that execute on the VXML Gateway in order to invoke the listed scripts.**

2. **VoiceXML Property**. Enter the **VoiceXML properties** that should appear in the root document so they become the default properties for this application.

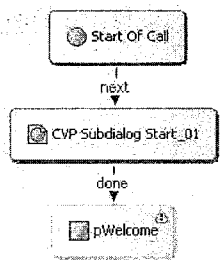
   - For example, interdigittimeout (DTMF timeout between touchtones), fetchtimeout (time gateway waits for audio or vxml to be returned), audiomaxage (use audio from the gateway cache if it's less than this many seconds old).

   - The list of Cisco supported VoiceXML Properties can be found in the ***Cisco VoiceXML Programming Manual.***

3. **VoiceXML Variable**. Enter VoiceXML variables to appear in the root document to ensure that they are available to all VoiceXML pages produced by the application.

   - **Note: Variables entered here requires custom Java voice elements that execute on the VXML Gateway in order to access the listed variables.**

## CVP Subdialog Start (Folder: Subdialog Elements/Cisco/)




1. It is recommended that all Studio applications begin with a **CVP_Subdialog_Start** element unless the application is only invoked from another Studio application.

2. Configuration of this element is **rarely** required. Except in these cases:

   A. If ICM passes data by setting 'UseVXMLParams' to 'Y' (not recommended) then you must configure the **CVP_Subdialog_Start** element to receive the parameters.

      a. **CAUTION** - If however UseVXMLParams was set to 'N' (as recommended by Cisco), then do **NOT** configure this Parameter setting.

      b. For each parameter that is passed as a **VXML Parameter** to the application from ICM, the "**Parameter**" setting must be configured with the name of the data that is passed. To add another Parameter, right-click and select Add Parameter.

      c. For example, if you've configured ICM to use VXML Params:
         **user.microapp.UseVXMLParams = "Y"**
         And you've passed the following data:
         **user.microapp.ToExtVXML[1] = "language=Spanish;account=1234"**
         Then configure the Parameter settings as follows:

| Name | Value |
|---|---|
| +Parameter | **language** |
| +Parameter | **account** |
| Store As | Session Data |
| *Enable Digits Bypass | false |

      d. The "**Store As**" setting allows you to choose to store all the passed values as Session data or as Element data. The name of the data will match the name of the parameter passed.

      e. **CAUTION**: If ICM passes a variable as a VXML Parameter, then it must be specified in the **CVP_Subdialog_Start** element.

   B. The "**Enable Digits Bypass**" setting is used to activate a VoiceXML workaround to ensure expected functionality **for a particular TDM** or analog phone and is set false for IP phones. If this setting is set to "true", the "Audio Filler URI" setting can be configured to set a reference to the audio containing silence "flash: silence_.1s.wav" file.

## Audio Element (Folder: Elements/)

1. The **Audio element** is used to speak greetings, error messages, hold messages, and any audio that is played but is not associated with collecting input.

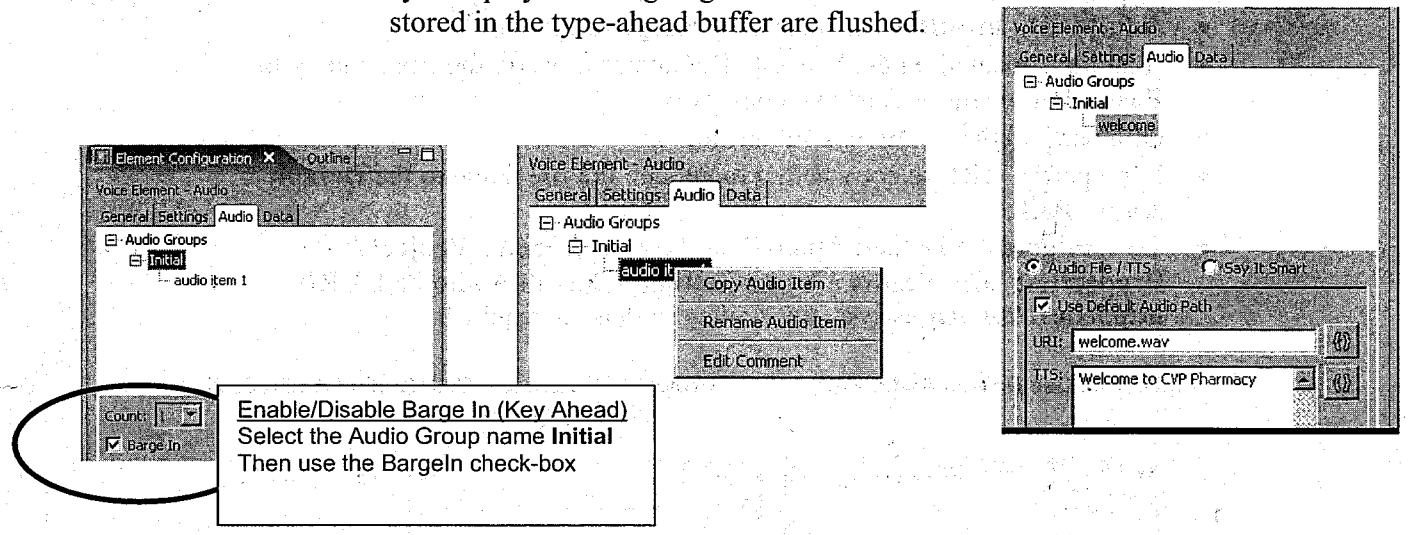
2. At runtime, the **Audio** element creates one VoiceXML page with the contents of the audio group called **Initial**. The **Initial** audio group may contain multiple audio items. Each audio item represents one prompt or audio file spoken to the caller.

3. To configure the **Audio tab** of the Audio Element:

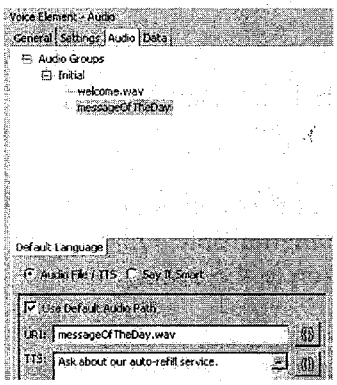
   A. Select the element in the Callflow Editor, and then click the Audio tab in the configuration pane.

   B. Expand **+Audio Groups**
   **+Initial**

   C. Select **Initial**
      a) You can enable/disable **Barge In** ("key ahead") for the Audio Group.
      b) If **Barge In** is enabled (the default), any DTMF or speech entered by the caller aborts this and successive prompts and is used as input to the next element that collects caller input.
      c) If **Barge In** is disabled, touch tones and speech input are ignored while the system plays the outgoing audio. Touchtones entered previously that are stored in the type-ahead buffer are flushed.




   D. Expand +**Initial**, and highlight the item named **audio item 1.** Audio Items are explained below:
      - Each Audio Group (for example, **Initial**) can be composed of one or more Audio Items (individual prompts).




**Audio Items**

1. **To add/delete** an audio item, **right-click** the **Initial** audio group and select Add/Delete Audio Item.
2. **To rename** the audio item, right-click and select RENAME (use alphanumerics, spaces, underscores).

**Item names can not contain the reserved symbols: apostrophe ', double-quote ", ampersand &, less than < , or greater than >.**

3. **To re-order** 2 audio items, drag one item above or below the other.
4. **Add Comments** to an audio item and they appear in the Documenter output
5. **Copy/Paste** items within or between different Audio Groups

E. Note that every audio item must be configured using **Pre-Recorded Audio, TTS, or Say it Smart**:

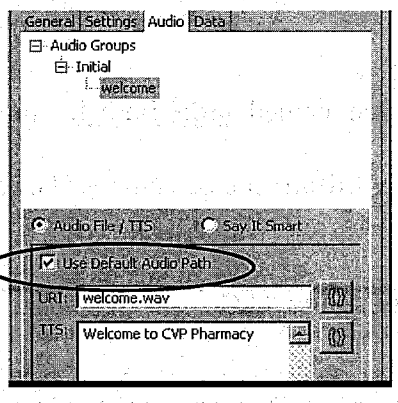
I. **Pre-Recorded Audio**

a) Enter a URI to speak pre-recorded audio retrievable by the gateway from the media server or flash.

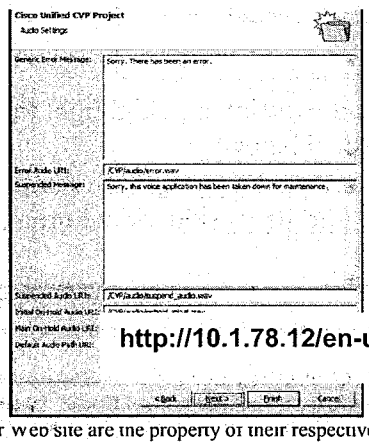
b) A URI can consist of

- A full URI: **http://myserver.com/prompts/hello.wav**
- If audio is located on the VoiceXML Gateway, prefix the file name with **flash:** For example, flash:welcome.wav
- Or a partial URI **helloworld_audio.wav**
- For a partial URI, select whether to append the file name to the **Default Audio Path**.
- To configure the Default Audio Path, follow the menu **Project / Properties: CallStudio / Audio Settings** Default Audio Path URI: **http://10.1.78.12/en-us/sys/** (the trailing slash is not required)

**Audio Tab, Use Default Audio Path**




**Project / Properties – Call Studio / Audio Settings Setting the Default Audio Path URI:**



II. **Text-to-Speech**

a) To use 3rd party Text-to-Speech synthesis to speak prompts, enter text in the TTS text box. At run time you must have the gateway configured to connect to the TTS server.
**Welcome to CVP Pharmacy at 12 Main Street, Waco, Texas.**

b) If both the URI and TTS are configured, the TTS is played only when the audio URI is not found.

c) TTS can contain markup tags, such as the <break/> tag to insert a pause. Any of the following tags are valid:

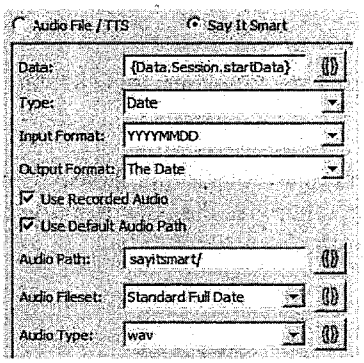
```
<break/>
<break  time="2s"/>
<break  time="500ms"/>
```

**Would you like to hear how you can save $100 per month?**
**<break/> If not, then please enter your account number.**

III. **Say it Smart** is used to specify that you'd like to speak the contents of a variable in a certain format. For example, speak as *digits, date, time*, *currency*, etc.

a) At run time, VXML Server will convert data into a list of audio file names and backup TTS to be spoken to the caller. This will be discussed later in the class.

b) For a list of required audio files, see the ***Say it Smart Specifications*** reference manual.



## CVP Subdialog Return (Folder: Subdialog Elements/Cisco/)

1. In a CVP Comprehensive Environment, you must use the **CVP Subdialog Return** element to return call control back to ICM to transfer the call, change call type, release the call, etc.

2. Configure the **CVP Subdialog Return** element with data to pass back to populate ICM's ECC variables:
   **User.microapp.caller_input**
   **User.microapp.FromExtVXML[0]**
   **User.microapp.FromExtVXML[1]**
   **User.microapp.FromExtVXML[2]**
   **User.microapp.FromExtVXML[3]**

3. The data must conform to the size of the ICM ECC variables configured on your ICM system (max 210 characters each, but yours may be smaller).

   - The **Caller Input** setting must be assigned a value. This value will be returned into ICM's **User.microapp.caller_input** variable.
   - The **External VXML 0,1,2,3** settings are optional and will be returned into ICM's **User.microapp.FromExtVXML[0], [1], [2], [3]** ECC variables.
   - The format of these entries is user-defined and should conform to whatever the ICM routing script expects.

4. In a **CVP Standalone environment**:
   a. The CVP Subdialog Return element is not required but can be used to end the call.

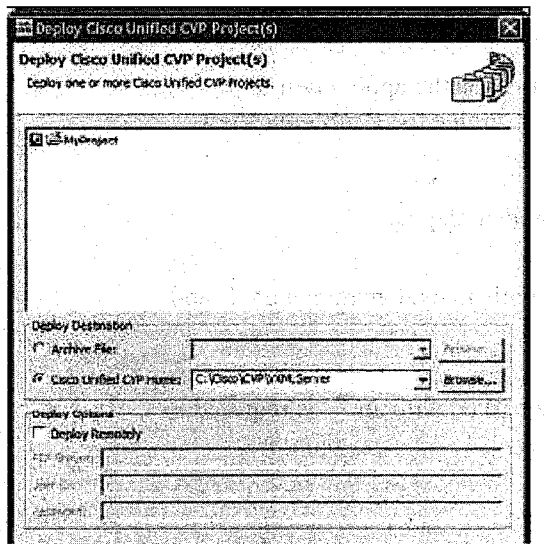
## Hangup

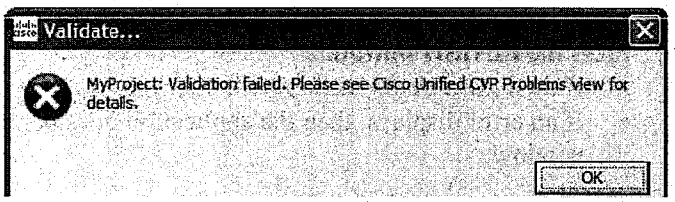
1. In a **CVP Standalone** environment, you may use this element to hang up the call and free up the VXML Server port license.

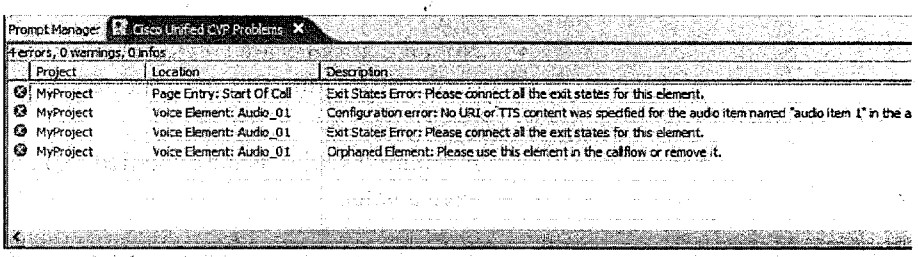
2. Do **NOT** use the Hangup element in a CVP Comprehensive environment if ICM has routed the call to the Studio application, otherwise the ICM Routing Script will follow the Run External Script node's **failure path**.

## Saving and Validating the Application

1. Save your application from **File > Save** (or Ctrl+s)
2. You may **validate** the application's configuration before deploying it. Validating is faster than deploying for a large application, but finds the same errors.
3. To validate, **right-click** the project in the navigator pane, choose **Validate**.
4. If there are errors, click **OK** and then select the **Call Studio Problems** window at the bottom of the screen to view the error.
   - Within the Problems View, **double-click the Location field** of the error message for Call Studio to highlight the Element causing the problem.




5. **NOTE: Validation (or Studio Deployment) doesn't find all errors. Some errors may not be obvious until runtime.**

## Deploying the Application (Creating the runtime code)

1. **Deploying** is similar to **compiling** the code. It creates the runtime files and stores them wherever you specify.

2. You must **SAVE** the application before deploying.

3. **Right-click** the project in the Navigator, choose **Deploy** (similar to "compile").

4. Applications can be deployed in 3 ways:
   a) **Archive** – creates a zip file for deploying with the **Operations Console**
   b) **Local** (or mapped drive) - select **C:/Cisco/CVP/ VXMLServer**
   c) **FTP** – deploy to an FTP server

5. In the classroom, select the local directory **C:/Cisco/CVP/VXMLServer**

6. Press the **FINISH button.**
   - If the Finish button is disabled, then you probably forgot to **save** the application first.
   - If an error displays, then the application does not deploy. Examine the **Problems View** window.

7. **Debugging Tips**: Common reasons an application will not validate/deploy:

   - Not all exit states are connected – any element with an exclamation sign ! and outlined in yellow has unconnected exit states.
   - A required setting is not specified. Any setting marked with an * is required to have a value.
   - An audio item has no URI or TTS specified.
   - A setting value is invalid (for example, you've specified a character value for an integer-only setting)
   - If your Finish button is disabled, you probably need to save the application.

## Administrative Scripts on the VXML Server

1. After Saving and Deploying from Studio, you must **always** run an administrative script on VXML Server. (This is not required when deploying through the Operations Console).

- If this is a **new** application, run the **deployApp.bat script**.
- If this is a **modified** application, run the **updateApp.bat** script.

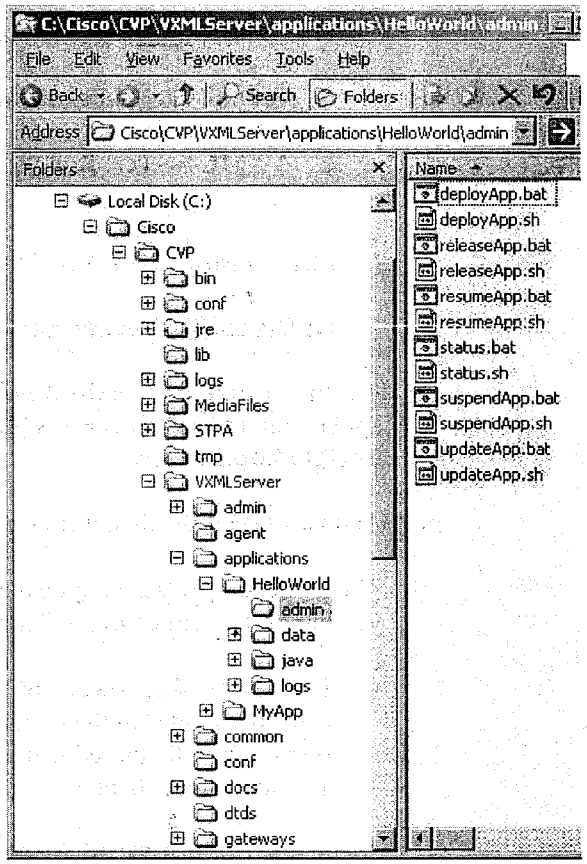
2. There are two levels of administrative scripts:
- **Global** – effect all VXML Server applications. Located in **C:/Cisco/VXMLServer/admin**

- **Application-specific** – effect only one application. Located in **C:/Cisco/CVP/VXMLServer/applications/<appname>/ admin**

3. To determine the number of current calls, license count, and valid application names double-click the global **status.bat** script, **C:/Cisco/VXMLServer/admin/status.bat**

4. The deployed application is stored in the folder: **C:/Cisco/CVP/VXMLServer/applications/<appname>/**

5. This folder contains four subfolders:
   - **admin** – administrative scripts for this application
   - **data** – the actual application
   - **java** –custom java to be used solely by this application
   - **logs**- application-specific log files will be stored here




6. To inform VXML Server about a new application, you must **double-click** the **deployApp**.bat script located in:

**C:/Cisco/CVP/VXMLServer/applications/MyApp/admin/deployApp.bat**

- This script instructs VXML Server to add the application name to its master list of available applications and also to load the application into memory.

- This step is not required if the application server (Tomcat or WebSphere) is not yet running or is restarted. It is also not required if you deploy using the Operations Console.

## Modifying an Application

If you make changes to a Call Studio application, you should do the following:

1. In **Call Studio:**
    a. **Studio**: Save the Call Studio application
    b. **Studio**: Right-click within the Navigator and select **Deploy**
    c. **VXML Server**: Run the **updateApp.bat** administrative script

2. In order to instruct CVP VXML Server about the update, run the administrative script:

    **C:/Cisco/CVP/VXMLServer/applications/MyApp/admin/updateApp.bat**

3. It is incorrect to run the **deployApp.bat** script again. It is only run once, when the application is new to CVP VXML Server. Each modification requires running the **updateApp.bat** script.

CAUTION - If you see this, it is as error message: *That application has already been deployed.*

4. The **updateApp.bat** script executes **gracefully**. All current callers experience the old version of the application. All new callers will experience the new version.
    - The **updateApp.bat** script will warn you of existing callers using the old version and it will 'countdown' in the CMD window, displaying the number of callers still using the old version. It will automatically display the new count each time the number of callers using the old version changes.

5. A global script **updateAllApps.bat** exists to update all applications at once. This is also a graceful administrative script located in
    **C:/Cisco/CVP/VXMLServer/admin/updateAllApps.bat**

6. The update scripts are **graceful** and do not effect current callers, only new callers.

## VXML Server "Global" Admininstrative Scripts C:/Cisco/CVP/VXMLServer/admin/

| Name (Arguments) | Time Frame | Description |
|---|---|---|
| `suspendServer (noconfirm)` | *Immediate, but Graceful* | Suspends CVP VXML Server. While the suspend goes into effect immediately (meaning all new callers hear the suspended message for their application), all **existing calls to applications will continue normally.** |
| `resumeServer (noconfirm)` | *Immediate* | Restores the status of each application to what they were at the time CVP VXML Server was suspended. So if an application was previously suspended, it will remain suspended. |
| `deployAllNew Apps (noconfirm)` | *Immediate* | All appls in the `applications` folder that are not in VXML Server memory are now loaded into memory and can handle calls. |
| `flushAllOld Apps (noconfirm)` | *Graceful* | When called, all apps in XML Server memory whose folders no longer exist in the `applications` folder are removed from memory. An application is not removed from memory until all callers leave it. While waiting for existing callers to hang up, the application is suspended to prevent new callers from entering. |
| `updateAllApps (noconfirm)` | *Graceful* | Performs an update on each application deployed on CVP VXML Server. |
| `updateCommon Classes (noconfirm)` | *Immediate* | Reloads all Java found in the `common` directory of AUDIUM_HOME. |
| `status` | *Immediate* | Displays information on VXML Server (such as the total callers visiting the system) and each application deployed on it (such as the number of callers interacting with the application). |
| `getVersions` | *Immediate* | Displays the version number of all VXML Server components. |
| `importLogs` | *Immediate* | Imports the activity logs of an application into a user-specified database. Uses a property file to load all appropriate information. |

## VXML Server "Application-Specific" Admin Scripts

C:/Cisco/CVP/VXMLServer/applications/<appname>/admin/

| Name (Arguments) | Time Frame | Description |
|---|---|---|
| `suspendApp (noconfirm)` | *Graceful* | Suspends the application in which this script resides. |
| `resumeApp (noconfirm)` | *Immediate* | Resumes the application in which this script resides. |
| `deployApp (noconfirm)` | *Immediate* | Load into memory the application in which this script resides |
| `updateApp (noconfirm)` | *Graceful* | Reload into memory all information about the application in which this script resides. |
| `releaseApp (noconfirm)` | *Graceful* | Remove the application in which this script resides from memory to prepare for its physical deletion. Use this option for Microsoft Windows systems where the application folder cannot be deleted until it's log files are closed. |
| `status` | *Immediate* | Displays information on the application in which the script resides such as the number of callers visiting it and whether there are callers experiencing an old configuration of the application. |

## Time Saving Tips:

1. By adding the '**noconfirm**' parameter to a shortcut to the **updateApp.bat and updateAllApps.bat** command, you will not need to confirm the script each time you run it. It is best to add this to the end of the command used in a shortcut to the script:

   **updateApp.bat noconfirm**

2. If you would like to suppress displaying the count down of callers using the old version of an application when you perform an update, you may add the '**nocountdown**' parameter to a shortcut to the **updateApp.bat and updateAllApps.bat** command:

   **updateApp.bat noconfirm nocountdown**

3. You can change the Activity Log rotation properties to create **one log file per day**, or per call, or by size (and app update) (default).

   a. Then use freeware, such as **tail.exe** to view the Activity logs and Error logs in real time.

   b. Use tail.exe to configure keywords (**Settings / Keywords**) highlighted and color coded to make it easier to find errors.

   c. For example, some good keywords to add are: **error, start** (start of call), **data** (variables that are created)

## Changing Activity Log Properties

1. By default, activity logs are rotated when they reach 50KB or each time the **updateApp**.bat script is run. This can be modified to rotate by time instead.
2. Logs can also be configured to be deleted automatically by the VXML Server.
3. Configuration changes can be made in the Call Studio application. When the application is deployed and the updateApp script is run, the new configuration will be instantiated.
4. In Call Studio open the **Project / Properties** window for a specific application.
5. Open the **CallStudio / General Settings** window
   - A. In the Loggers box, select the **ActivityLog**
   - B. Press **Edit**
     - i. The Logger Instance window will display. Press **Edit**
     - ii. The **ActivityLogConfig.xml** file displays, edit as follows and press **OK.**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration SYSTEM "../../../../dtds/ActivityLoggerConfig.dtd">
<configuration version="1.0" name="ActivityLog">
<format delimiter="," remove_delimiter_from_content="true" date_format="standard" date_granularity="milliseconds"/>
<scope logging_level="Complete">
<definitions>
<level name="Complete">
<event id="start"/>
<event id="end"/>
<event id="elementEnter"/>
<event id="elementExit"/>
<event id="elementFlag"/>
<event id="defaultInteraction"/>
<event id="elementData"/>
<event id="custom"/>
<event id="hotlink"/>
<event id="hotevent"/>
<event id="warning"/>
<event id="systemError"/>
<event id="javaApiError"/>
<event id="xmlApiError"/>
<event id="vxmlError"/>
</level>
</definitions>
</scope>
<rotation>
<by_size mb_limit="100"/>
</rotation>
<cache>
<per_call kb_limit="50" allocate="once"/>
</cache>

<purge>
<file_age older_than="31"/>
</purge>

</configuration>
```

Rotation options - use one of the following, replacing the number with your own number.

a. **<by_day every="1"/>**
b. **<by_hour every="12"/>**
c. **<by_call/>**
d. **<by_size mb_limit="100"/>**

For auto-purge by VXMLServer, add one of these entries (in bold) right before the last line of the file:

a.

```
<purge>
<file_age older_than="7"/>    (days)
</purge>
```

b.

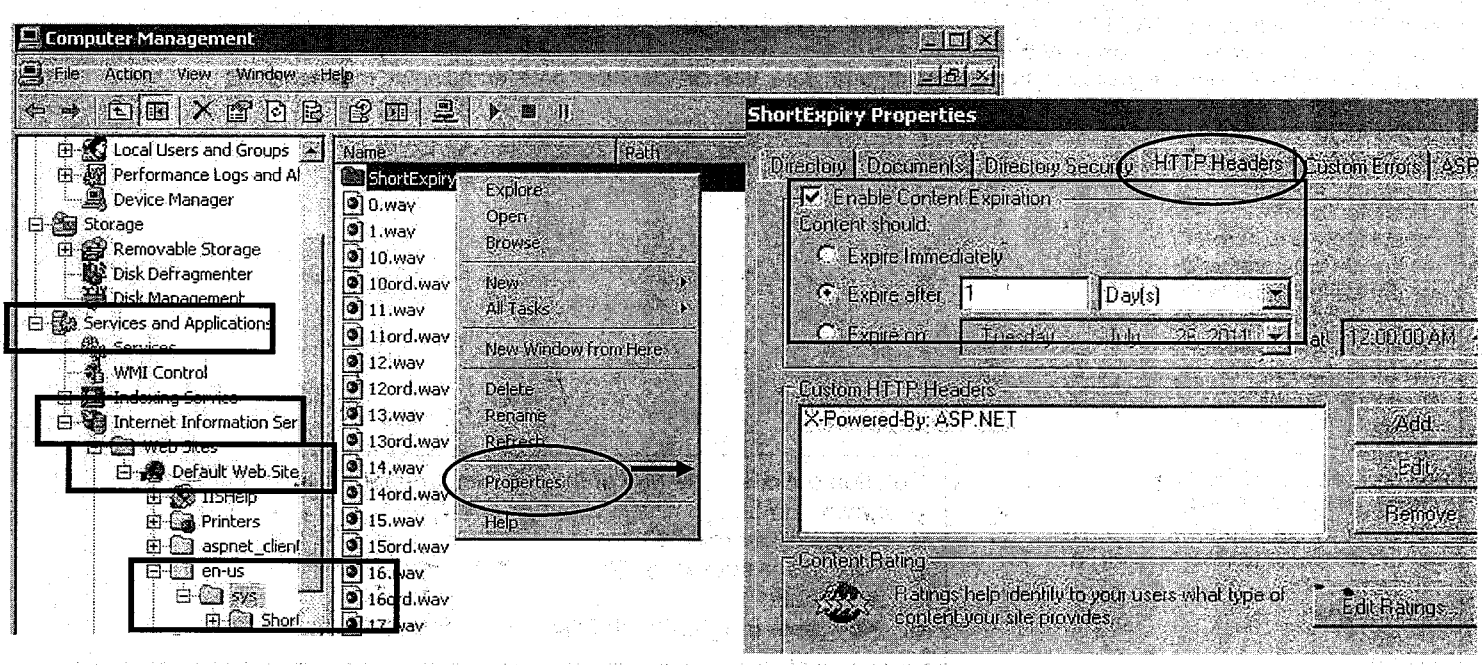
```
<purge>
<file_count greater_than="5"/>
</purge>
```

## Understanding Cisco Audio Caching (Reference)

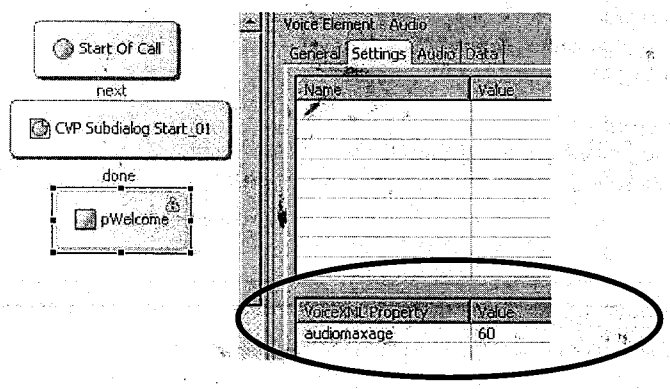
1. Audio files retrieved from a Media Server are cached on the VoiceXML Gateway to improve performance. Note that if a file is not cached then that audio isn't shared by callers and requires extra memory for each caller hearing it. And when the gateway is done playing it, that file is deleted from the gateway immediately.

2. Files are retrieved and cached until they 'expire'. On an IIS Media Server, you can configure **the cache expiration for each directory** or **file** to a value that allows re-recorded audio files to replace their predecessor in a reasonable amount of time, while minimizing requests for data to the media server from the VXML Gateway. Note that if the media server file hasn't changed, it will not be downloaded even after it's expired.

**Start > MyComputer (Right-click) Manage**




3. Once in the cache, an audio is used from the cache until it either becomes **'stale'**, i.e., its expiration time as configured on the Media Server (or the gateway) has expired; or the Studio application has configured an **audiomaxage** smaller than the length of time the audio's been in the cache.



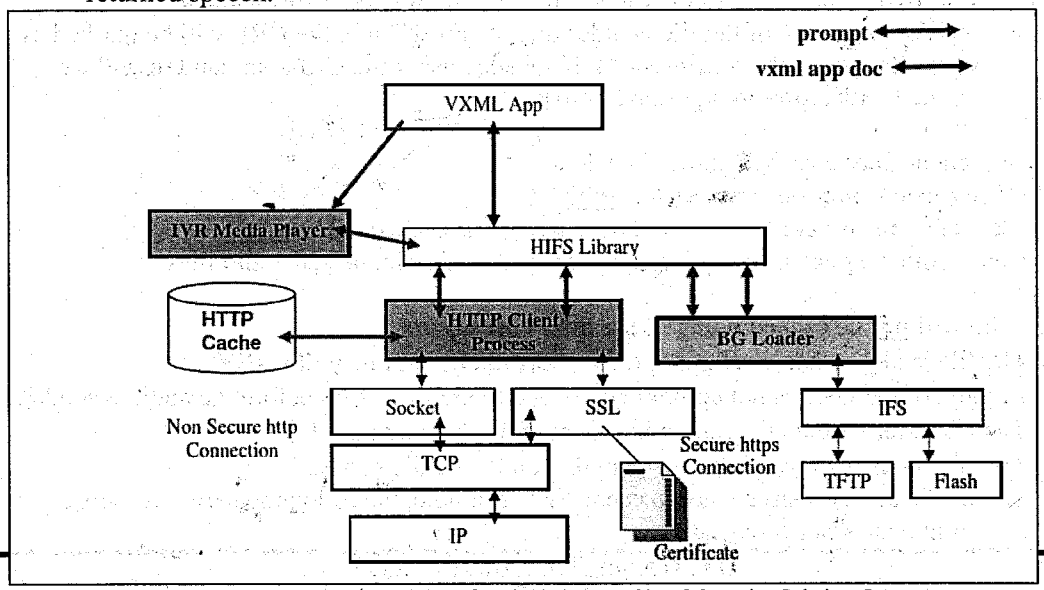

**audiomaxage**
Enter an integer value (representing seconds)

4. The order of precedence of these settings is
   - VXML application settings have highest priority (**audiomaxage, audiomaxstale**)
   - The media server setting take second highest priority (HTTP Header Expiration)
   - The gateway configuration has lowest priority (**http client cache refresh <seconds>**)

5. If an audio file in the cache is not **'stale'** then the ONLY ways to force the gateway to 'reload' or 'refresh' it immediately in the cache is:
   - Modify the **Studio application** to set theVoiceXML property **audiomaxage** to a small value, call into the application through each gateway.
   - Use the Gateway command: **audio load <URL>** on each gateway for each file needing a refresh. Note that RefCount must be zero in the **show http client cache** display for this command to succeed.
   - Set all the entries in the http client cache as stale using the gateway command **set http client cache stale.** This sets the entire cache stale and causes an http "conditional-**GET**" including an "If Modified-Since" in the headers for each audio URI as it's requested to be spoken. If it hasn't changed, the system doesn't reload this audio file, but uses it from the cache, and leaves it marked stale in the cache.
   - Reload or restart the gateway. **Warning** - this adversely impacts current calls.

**The remainder of this discussion is for reference only**

1. The gateway has many asynchronous processes handling audio playback:
   - The **VXML App** (the voice browser) interprets the VXML code and hands off audio requests so it can continue processing the VXML code.
   - The **HIFS** process parses the URL and directs it to the HTTP Client; or to the BG (background) Loader for TFTP and Flash audio requests.
   - The **HTTP Client** is responsible for checking the **HTTP Client Cache** for the audio file, retrieving the audio if necessary, and storing it in the cache.
   - The **IVR Media Player** plays the audio file to the caller.after the gateway downloads it.
   - **MRCP Client** interfaces to the Text-to-Speech Server sending text and playing out the returned speech.



2. Gateway Caching Commands:

A. **http client cache memory pool <0-100000>** Configure total cache memory size (Kbytes). 0 to disable caching; Recommended 1000K (10M)

B. **http client cache memory file <1-10000>** (K kilobytes). Any single file larger than this configuration setting will **not** be cached. Default 50K. Recommended 600K.

C. **show http client cache** – CLI command: Displays the contents of the gateway cache, shown below:

D. **set http client cache stale -** CLI command: sets entire cache stale, forcing an http "conditional" **GET** on the next speak request of each audio URI, only reloading a file if the Media Server file is different from that in the cache.

E. **clear http client cache** (no longer supported) - CLI command: Clears all audio from the cache, forcing a reload on the next speak request of each audio URI. This has been deprecated, but still works.

**HTTP Client cached information**

Maximum memory pool allowed for HTTP Client caching = 20000 K-bytes
Maximum file size allowed for caching = 400 K-bytes
Total memory used up for Cache = 3558 Bytes  Message response timeout = 10 secs
Total cached entries = 4  Total non-cached entries = 1

Cached entries

| Ref | FreshTime | Age | Size | Context |
|---|---|---|---|---|
| 2 | 86400 | 126358 * | 465 | 67518CDC |

**url: http://10.1.78.72/en-us/sys/welcome.wav**

- **Ref** is number of calls currently using this URL. **No URLs can be removed from the cached entry until its ref count is zero.**
- **Age** is the elapsed time since the file was last downloaded from the media server.
- **FreshTime** is the "life expectancy" (seconds). That is, the duration that the file is expected to stay fresh in the cache since the file was last downloaded.
- NOTE- If the age of the file is older than its FreshTime, the URL will be marked as **stale** (flagged with an **asterisk ***). If set stale using the CLI command, it will be marked with a **pound sign/hash mark #**

# STALE → POUND

3. The http client frees a cached audio file when:
   - The cached entry becomes **stale**; **AND**
   - Its **ref count** is zero (0), i.e., no one is using this cached entry; **AND**
   - Its **memory space** is needed to make room for other requested audio files.

4. An audio will **not** be cached if any of the following occurs:
   - The file is larger than configured **http client cache memory file <KB>**
   - Content of the URL is not completely loaded (caller hangs up before the audio is loaded)
   - The file is larger than the remaining memory in the cache pool.
   - The file on the gateway is newer than that on the media server.
   - The media server instructs not to cache. For example, the following server response header indicates not-to-cache:

**Cache-Control: no-cache**
**Cache-Control: private**
**Cache-Control: no-store**
**Cache-Control: max-age = 0**
**Pragma: no-cache**
**Expires: 0**

5. The cache is managed by each entry's "**freshness**". Whether a cached entry is **fresh** or **stale** depends on:
   - **Age** is the elapsed time since the file was last downloaded from the media server.
   - **FreshTime** is the duration that the file is expected to stay fresh in the cache since the file was last downloaded.

A. The FreshTime is calculated as follows: When a file is downloaded from the media server:
   - If the following is configured by the media server, then it is used as the Fresh Time:
     **Cache-Control: max-age = <value in seconds>**
   - Else, if the following 2 headers are configured then the difference is the Fresh time:
     **Expires: <Expiration date time>  Date: <Current date time>**
   - Else, the gateway calculates and uses the following for the FreshTime:
     **FreshTime = 10% x ( (LastModifiedDateTime) – (CurrentDateTime))**

C:\WINDOWS\SYSTEM32\LOGFILES\W3SVC1

DIRECTORY REQUEST 304 / NOT CHANGE

**Review Questions**

1. Identify the directory for each of the following:
   A. The source code for your Studio applications (This is important for backing up your applications)
      C:\CALL STUDIO\ECCIPSE\WORKSPACE
   B. The runtime code on the VXMLServer for your applications
      C:\CVP\VXMLSERVER\APPLICATIONS
   C. The detailed activity and error logs for the MyApp application
      C:\CISCO\CVP\VXMLSERVER\APPLICATIONS\ERROR LOG APP*
   D. The local administrative scripts that effect only MyApp
      C:\CISCO\CVP\VXMLSERVER\APPLICATIONS\ADMIN APP*
   E. Global administrative scripts for determining the number of licenses and callers into the VxmlServer
      C:\CISCO\CVP\VXMLSERVER\ADMIN
   F. The default audio files (example, error.wav, suspended_audio.wav) that install with VXMLServer C:\CISCO\CVP\VXMLSERVER\TOMCAT\WEBAPPS\CVPAUDIO
      C:\CISCO\CVP\MEDIAFILE\EN-US\SYS - SAYSMART
2. After you Save and Deploy from Studio. Which administrative script must you run on VXML Server? DEPLOYAPP.BAT
   UPDATEAPP.BAT
3. In Studio, where/how do you set the Default Audio Path? Which directory does this correspond to on the Media Server?
   PROPERTIES FOR MYAPP (AUDIO SETTINGS): DEFAULT AUDIO URI
   C:\INETPUB\WWWROOT\EN-US\SYS
4. In Studio, which nodes should you start and end with?
   CVP SUBDIALOG START ~~START OR BACK~~ CVP SUBDIALO RETURN
   ~~END OF CACE (HANG UP)~~
5. What are 3 ways to force the gateway to reload an audio file from the Media Server into its cache? Why might an audio fail to reload into the cache?
   SAVE YOUR STUDIO APP, SPEAK THE AUDIO FILE, AND SET VOICE PROPERTY NAME: AUDIOMAXAGE VALUE: 0 (ENTER REPRESENTING THE NUMBER OF SECONDS
6. In Studio, what are 3 reasons an application would fail to validate or deploy?

[End of Chapter 2]

5.2 - GATEWAY; AUDIO LOAD (URL)
MIGHT NOT WORK - IF THE AUDIO FILE IS IN USE (2-27)

5.3 - GATEWAY; SET HTTP CLIENT CACHE STALE

# Chapter 3

# Collecting Caller Input: Menu, Confirmations, Transferring the Call

- The Menu Elements
- YesNo Menu
- Where Collected Input is Stored
- Substitution Tag Builder to Access Variables
- Creating Multiple Pages
- Telephony Transfer Element
- Call Studio Debugger (Call Simulator)
- Using Activity and Error Logs to Debug
- Reference - Adding an Option to a Menu

## The Menu Elements (Folder: Elements/Menu/)

1. There are two ways to provide a menu. The **N_Option_Menu** elements which have a fixed number of options and branch the code; or the **Form** element which has unlimited options, but doesn't branch the code.

2. The **Menu** folder in the Elements pane contains voice elements for menus that support a fixed number of options with an exit state for each option.

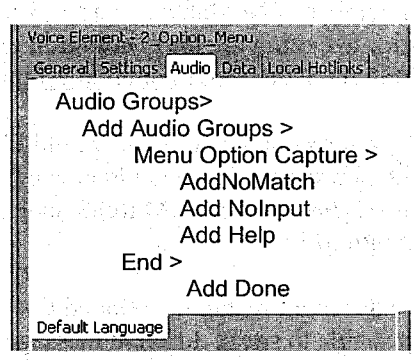
3. To configure the Menu element, use the **Audio tab** of the Menu element to specify the prompts. Use the **Settings tab** of the Menu element to configure input settings.

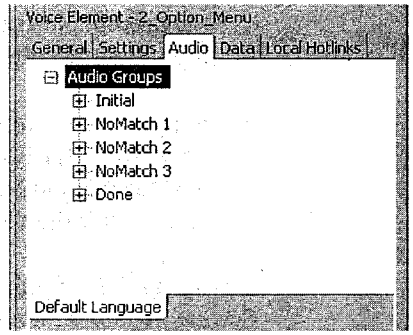
4. More details on using the **Menu** Element are in the *CVP Element Specification* reference manual.

## AUDIO

1. **Initial Audio Group** - The only required audio group is the **Initial** audio group which speaks to the caller upon entering the element, after which the system waits for caller input. If a caller input errors occur (no match or no input) the Initial prompt is replayed to the caller unless custom reprompting has been added.

2. **Custom Reprompting** - The **Audio Group name** and **number** determines when it's spoken. Each element gives the caller multiple tries to enter a valid entry. The default is to play the Initial Prompt before each retry. To add custom reprompting, **right-click** on **Audio Groups**.


| Add Audio Groups > | Audio Group | Description |
|---|---|---|
| **Menu Option Capture >** | **Initial** | Initial prompt spoken to the caller. System then waits for caller input. |
| | **NoMatch1,2,3** | Spoken if caller input doesn't match a menu option or if confidence is below the cutoff value. System then waits for caller input. |
| | **NoInput1,2,3** | Spoken to the caller if the NoInput timer expires. System then waits for caller input. |
| | **Help1,2,3** | Spoken to the caller if caller says 'help' or if the VXML event named 'help' occurs. System waits for caller input. |
| **End >** | **Done** | Spoken to the caller **after valid input is collected.** |

1. When you right-click an audio group name (e.g., ***Initial***), you can enable or disable barge-in for that group.

2. BargeIn allows caller input to abort the outgoing message, If BargeIn is disabled, caller input during the outgoing prompt is not retained.

3. Understanding Audio Groups

A. **Initial Audio Group** This tells the caller what to enter. **It is the only required audio group.**

B. **NoMatch Audio Group**– The **NoMatch** group is spoken if the caller's input doesn't match the required input or if the spoken utterance has too low a confidence.

- The NoMatch setting is **repeatable** allowing you to configure different messages to play upon successive nomatch events (NoMatch1, NoMatch2, NoMatch3).

- When a nomatch event occurs the system finds the group with the "**correct count**": the highest count among the groups (NoMatch#) that is less than or equal to the actual current counter value.

- So, if you configure **NoMatch1** and **NoMatch2** prompts, the caller hears **NoMatch1** on the first nomatch event, and then hears **NoMatch2** for the 2nd and all succeeding nomatch events while in this element.

- **Note** that if you configure the maximum number of NoMatch events as 3 (default) then the prompt **NoMatch3** will be spoken prior to exiting the **max_nomatch** exit state. No input will be collected. The prompt **NoMatch3** audio should not request input.

- Suppose you allow **eight** nomatch events (not recommended), but would like to specify only **NoMatch1, NoMatch2** and **NoMatch8** prompts. You should create **NoMatch1, NoMatch2** and **NoMatch3** prompts. Now **right-click** on **NoMatch3** audio group name, and modify the **Count** counter by changing it to 8.

C. **NoInput1,2,3** – Spoken if the caller doesn't enter input within the allotted time (see the Setting, *Noinput timeout*). This prompt is repeatable to speak different messages upon successive no input events.

D. **Help1, 2, 3** - Spoken if the caller requests '**help**'. This setting is repeatable to speak **different** help messages upon successive help requests. By default, 'help' is enabled for speech recognition systems using a built in recognition grammar containing the word 'help' To enable 'help' using DTMF, use Hotlinks

**E. Done** – Spoken only after caller input is **successfully** captured. At this point, the element data {**Data.Element.*MenuElementName*.value**} has been created and contains the setting **OptionN Value** for the caller's selection.

**Note:** You can right-click an audio item and select Copy/Paste, Rename, Edit Comment, and set the language (for TTS).

**CAUTION: Audio item names can not contain the characters: < > " ' or &**

**Classroom Example:**

**Initial Prompt:**
**Audio item1** *What would you like to do? You can refill a prescription or use our store locator to find a pharmacy.*
**NoMatch1 Prompt:**
**Audio item1** *I didn't understand.*
**Audio item2** *For refills, say refill a prescription or press 1. To locate a pharmacy, say store locator or press 2.*
**NoMatch2 Prompt:**
**Audio item1** *I still didn't understand.*
**Audio item2** *To refill a prescription press 1. To find a pharmacy, press 2. Or stay on the line for a customer care representative.*
**NoMatch3 Prompt:**
**Audio item1** *I just can't understand.*
**NoInput1 Prompt:**
**Audio item1** *I didn't hear you.*
**Audio item2** *For refills, say refill a prescription or press 1. To locate a pharmacy, say store locator or press 2.*
**NoInput2 Prompt:**
**Audio item1** *I still didn't hear you.*
**Audio item2** *To refill a prescription press 1. To find a pharmacy, press 2. Or stay on the line for a customer care representative.*
**NoInput3 Prompt:**
**Audio item1** *I just can't hear you.*
**Done Prompt:**
**Audio item1** *You've selected to*
**Audio item2** <insert the variable that represents the caller's selection>

## Menu Element Settings

1. Settings must be configured for the **Menu** element to function properly.
   A. Settings with *** are required** to have a value.
   B. Settings with **+ are repeatable**, you may right-click and select Add Another

| Settings | |
|---|---|
| ***Noinput Timeout** | The maximum time allowed for silence or no keypress before a **NoInput** event is thrown. Use a time unit (s for seconds, ms for milliseconds) |
| ***Max NoInput Count** | The maximum number of noinput events allowed during collection. |
| ***Max NoMatch Count** | The maximum number of nomatch events allowed during collection. |
| ***Confidence Level** | The confidence level threshold to use during collection. Results with confidence below this return as a **nomatch** event. (DTMF entries are assigned confidence 1.0). Default 0.5 |
| ***Disable Hotlinks** | Whether to disable all hotlinks (all other active grammars besides what this element specifically collects) |
| **+Option X Voice** | A word or phrase the caller can say to trigger that path of the menu.<br>1. To include synonyms, Right-click and select *Add Another Voice.*<br>2. Leave blank if you are not using ASR.<br>**3. Either a Voice or DTMF setting must be specified for each option.**<br>4. This setting is repeatable.<br>**5. DO NOT SPECIFY A VALUE HERE IF USING THE DTMF_ONLY GATEWAY ADAPTER.** |
| **+Option X DTMF** | A 1-digit DTMF keypress to send the call down that menu path. Use 0-9, *, or # (must disable termchar to collect # from the caller).<br>1. Leave blank if you are not using DTMF.<br>2. At least one of Voice or DTMF setting must be specified for each option.<br>3. This setting is repeatable. |
| **Option X Value** | Specify a value to return into the Element data named 'value' when the caller selects this menu path. |

### Classroom Example Settings:

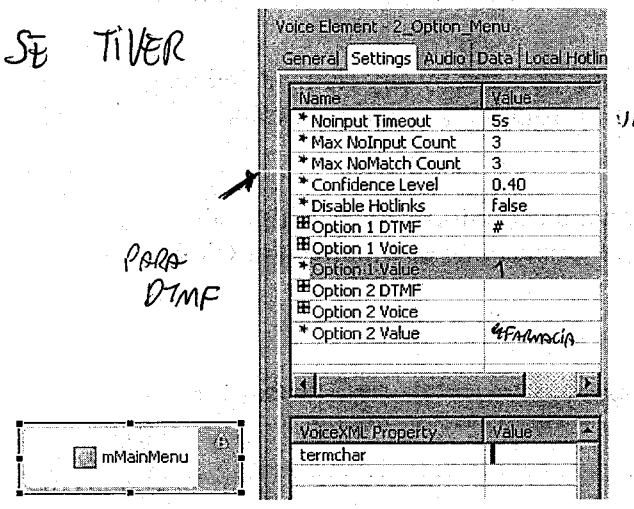
| | |
|---|---|
| **Option1 DTMF** | **1** |
| +Option 1 Voice | refills |
| +Option 1 Voice | refill a prescription |
| *Option 1 Value | refill a prescription |

| | |
|---|---|
| **Option1 DTMF** | **2** |
| +Option 1 Voice | find a pharmacy |
| +Option 1 Voice | find a store |
| +Option 1 Voice | store locator |
| *Option 1 Value | find a pharmacy |

## Menu Settings: VoiceXML Property Settings

A. The **Settings** tab allows you to enter **VoiceXML properties** to be used for this element only. VoiceXML properties effect caller input, http fetch timeouts, caching, audiomaxage, etc. For a list of supported VoiceXML properties, see the *Cisco VXML Programming Manual*.

B. Example: To collect # as a menu entry, you must temporarily disable # from being used as the terminating character and stripped off by the gateway.




a. Set the VoiceXML Property name to **termchar**.

b. Set the Value to one **space**.

c. This allows # to be collected from the caller and sent to the VXML Server application during this one element only.

d. After this element is completed, # will again become the termination character allowing callers to indicate the end of a multi-digit entry.

## Menu Element Exit States

Voice collection elements have multiple exit states that must be connected.

| Exit States | |
|---|---|
| **option1** | The utterance or DTMF entry triggered Option 1 |
| **option2, option3, etc.** | The utterance or DTMF entry triggered Option 2 (or Option 3, etc.) |
| **max_nomatch** | The maximum number of nomatch events has occurred. |
| **max_noinput** | The maximum number of noinput events has occurred. |

## Menu Element Variables

Voice collection elements store the caller's input into the following **Element data** variables that belong to the element that created the data and are referenced as **{Data.Element.*ElementName.VariableName*}**

| Element Data Created | |
|---|---|
| **value** | A representation of what the caller entered as specified in the setting named OptionN Value |
| **value_confidence** | The confidence value returned by the ASR server. This is a float value between 0.0-1.0 representing how closely the caller's input matched something in the grammar. (DTMF value_confidence always has the value 1.0) |

## Substitution (Using Variables)

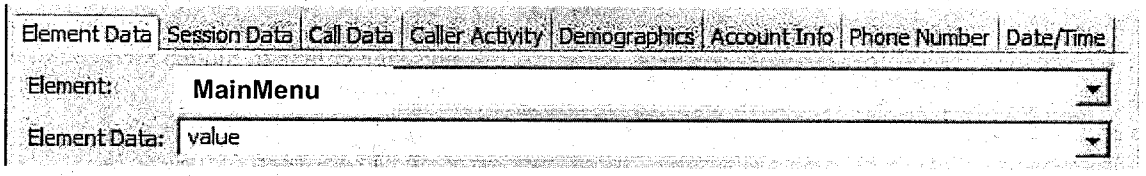
1. To access variables, select the **Substitution Tag Builder** button { } located in the Configuration view of any element.

2. Call Studio specifies the variables to insert by means of a **Substitution tag** built using the **Substitution Tag Builder**.

   A. An **Element data** variable is created by an Element and 'belongs' to that Element (no other element may change its value). It may be treated as Float, Integer, Boolean or String values.

   B. A **Session data** variable can be created and changed by any element and may exist at the start of the Session if passed in from ICM or another Studio Application.

   C. **Call data** variables are read-only and contain call information such as call source, application name, and in the CVP Standalone environment or CVP8 the ani and dnis.




3. Within the tag builder:

- Select the tab at the top named '**Element Data**'
- Then select the element name that collected the input. Element: **MainMenu**
- Then select the variable name Element Data: **value**
- Then press the button '**Add Tag**'
- Then press the button '**OK**'

4. Element data variables are identified by the **Element name and the Variable name.** The substitution tag for element data looks like this:
{**Data.Element.<*ElementName*>.<*VariableName*>**}

**{Data.Element.mMainMenu.value}**

5. **WARNING:** Before CVP8 changing the name of the original element required you to manually change all references to its Element data throughout the application.

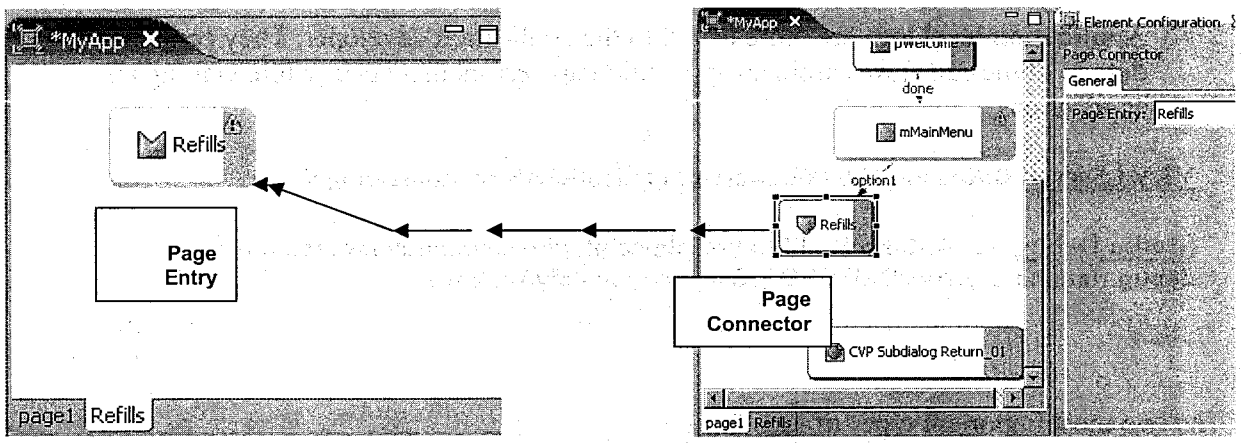
## Creating Multiple Pages

1. Large-scale voice applications have call flows that are too large to fit on a single page. Call Studio provides the ability to define an application on as many pages as desired.

2. To create a new page, **Right-click** the **page1** tab at the bottom of the Callflow Editor and select **New Page**.

   A. To rename a page, **Right-click** the **pageX** tab and select **Rename Page**.

   B. Each page you create has a **Page Entry** named **pageX** which can be renamed to match the page name.




3. To connect the call flow across pages use a **Page Connector** to send the call flow to a **Page Entry** with the selected name.

4. Drag a **Page Connector** from the Element toolkit into the callflow at the point where you wish the callflow to connect to the new page.

5. To configure the Page Connector, pull-down the menu in the Page Connector's configuration pane. This menu displays all possible Page Entry elements. Select one.

   - First create the Page Entry and name it.
   - Then create the Page Connector and configure it with the name of the Page Entry to 'connect to'.

6. **Page Entries** and **Page Connectors** can also be used to create a cleaner call flow within one page by reducing the number of lines within a single page. This allows for easier movement of code later on.

7. **Reorder Pages**: You may reorder pages by dragging the page name tab at the bottom of the screen to the desired location.

## Debugging Your Application Using Log Files

1. After you Save and Deploy your application from Studio, and then run the updateApp.bat script, if your application ends unexpectedly, it is usually due to an unexpected error. The best way to debug is to look in the logs.

2. VXML Server has two levels of logging: **global logs** and **application level logs**.

- **Global logs** are used by administrators to monitor the number of calls into the system and errors that aren't associated with an application.
  **C:/Cisco/CVP/VXMLServer/logs/**

- **Application level logs** are more useful to the application developer. They contain **details** of the call flow within an application and errors that occur while visiting an application.

**VXMLServer** **C:/Cisco/CVP/VXMLServer/applications/<appname>/logs/**

**Studio Debugger**: **C:\Cisco\CallStudio\eclipse\plugins\com.audiumcorp.studio.debug.runtime_x.y.z\AUDIUM_HOME\applications\MyApp\logs**

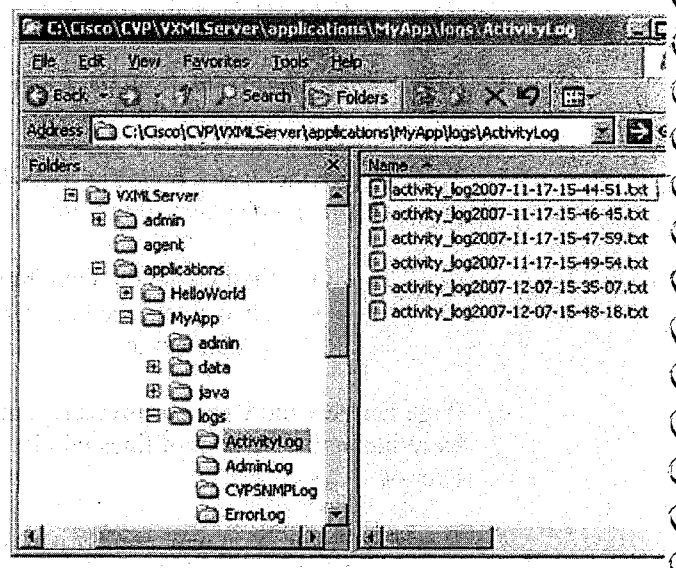
3. The two most important types of logs for the application developer are the following application level logs:

- **Activity logs** contain detailed call flow information including the name of each element executed, the exit state followed, the information collected from the caller, the source of the call, how the call ended, duration of the call.

  **C:/Cisco/CVP/VXMLServer/applications/<appname>/logs/ActivityLog/**

- **Error logs** contain detailed information about errors that occur.

  **C:/Cisco/CVP/VXMLServer/applications/<appname>/logs/ErrorLog/**



## Application Level Activity Logs

**VXML Server Call ID** **Date** **Time** **Element** **Event** **Value**

Session ID

Source of the call

1. **10.1.78.12.1194352167109.0.MyApp**,11/06/2007 07:29:27,,**start,newcall,**
2. 10.1.78.12.1194352167109.0.MyApp,11/06/2007 07:29:27,,start,ani,4002
3. 10.1.78.12.1194352167109.0.MyApp,11/06/2007 07:29:27,,start,areacode,NA
4. 10.1.78.12.1194352167109.0.MyApp,11/06/2007 07:29:27,,start,exchange,NA
5. 10.1.78.12.1194352167109.0.MyApp,11/06/2007 07:29:27,,start,dnis,2002
6. 10.1.78.12.1194352167109.0.MyApp,11/06/2007 07:29:27,,start,iidigits,NA
7. 10.1.78.12.1194352167109.0.MyApp,11/06/2007 07:29:27,,start,uui,NA
8. 10.1.78.12.1194352167109.0.MyApp,11/06/2007 07:29:27,,start,**parameter,callid=987654ABC123**

Data passed in from ICM

9. MyApp,11/06/2007 07:29:27,Subdialog_Start_01,element,enter,
10. MyApp,11/06/2007 07:29:27,Subdialog_Start_01,element,exit,done

11. MyApp,11/06/2007 07:29:27,**p_hello**,element,enter,
12. MyApp,11/06/2007 07:29:27,p_hello,interaction,audio_group,initial_audio_group
13. MyApp,11/06/2007 07:29:27,p_hello,element,exit,done

Audio spoken and events that occurred on the gateway.

14. **07:39:25,get_rxnum,element,enter**
15. 07:39:25,get_rxnum,element,interaction,*audio_group,**initial_audio_group***
16. 07:39:25,get_rxnum,element, interaction,***nomatch,1***
17. 07:39:25,get_rxnum,element, interaction,audio_group,***nomatch_audio_group***
18. 07:39:25,get_rxnum,element, interaction,***nomatch,2***
19. **07:39:25,get**_rxnum,element, **interaction,audio_group,*nomatch_audio_group***
20. 07:39:39,get_rxnum,**data,*value,1234***
21. 07:39:39,get_rxnum,**data,confidence,***0.39***
22. 07:39:39,get_rxnum,**data,value_confidence**,0.39

Element data created

23. 07:39:25,get_rxnum,interaction,audio_group,initial_audio_group
24. 07:39:39,get_rxnum,interaction,**utterance,one two three four**
25. 07:39:39,get_rxnum,interaction,***inputmode,voice***
26. 07:39:39,get_rxnum,interaction,***interpretation,1234***
27. 07:39:39,get_rxnum,interaction,***confidence,0.39***
28. 07:39:39,get_rxnum,interaction,*audio_group,done_audio_group*

29. 07:39:39,get_rxnum,element,exit,***done***

Exit State Followed

30. MyApp,11/06/2007 07:39:48,Subdialog_Return_01,element,enter,
31. MyApp,11/06/2007 07:39:51,Subdialog_Return_01,element,exit,

32. MyApp,11/06/2007 07:39:51,,**end,how,hangup**
33. MyApp,11/06/2007 07:39:51,,**end,result,normal**
34. MyApp,11/06/2007 07:39:51,,end,duration,25

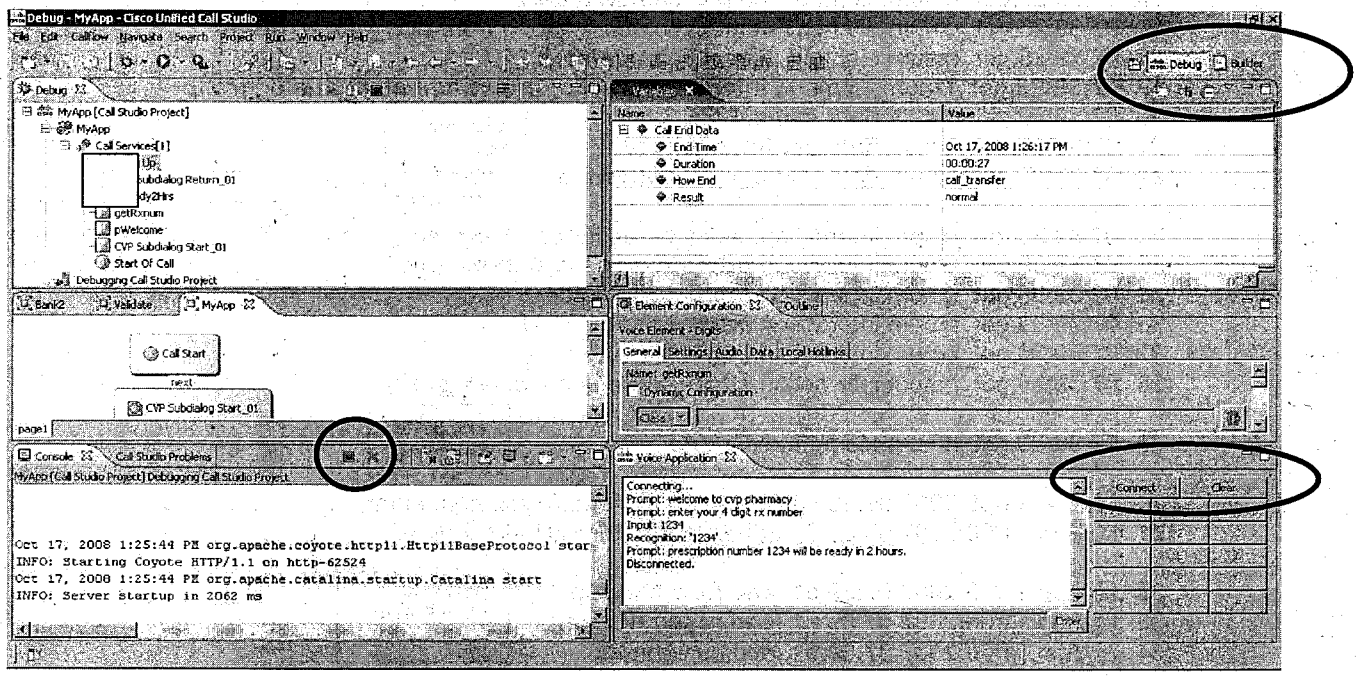
How Call Flow Ended

## Call Debugger

1. Call Studio includes a **Callflow Debugger**. The debugger starts an instance of Tomcat - VXML Server to run the application on the Call Studio workstation.

2. To start the Debugger, **right-click** the (validated) application in the Studio Navigator and select **Debug Call Studio Project.**

3. The Debugger window displays a number of '**Views**' (window panes):
   A) **Debug**: Elements that have executed (reverse-order) after 'placing a call'.
   B) **Variables:** Displays Session and Element data for an element selected in Debug view.
   C) **Console** The Console output (stdout.log) for Studio's Tomcat (VXMLServer Simulator).
   D) **Voice Application**: A touchtone telephone.
      - **Press Connect to start a call and Hang Up to end the call.**
      - The audio/TTS spoken to the caller displays in the box
      - Press the **DTMF** buttons or enter 0-1,*,# in the input box.
      - **NOTE - omit # as a terminating character.**
      - Enter caller utterances as text ("yes", "no") in the input box for voice input.
      - Press the **No Input/No Match** button to simulate these input errors.
4. When done debugging, you should click on the **red square Terminate** button.
5. To exit the Debugger view, press the **Builder** button in the upper/right corner.
6. The Debugger's log files are located in the directory:
   **C:\Cisco\CallStudio\eclipse\plugins\com.audiumcorp.studio.debug.runtime_x.x.x\AUDIUM_HOME\applications\<AppName>\logs**



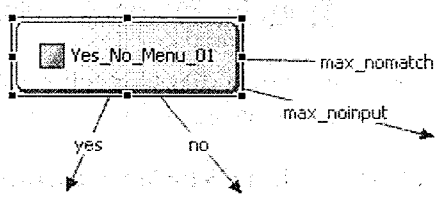







C:\Cisco\CallStudio\eclipse\plugins\com.audiumcorp.studio.debug

## YesNo_Menu (Folder: Elements/Menu/)




1. The **YesNo_Menu** presents the caller with a prompt and allows the caller to enter either DTMF **1 (for yes),** or **2 (for no)** or to speak their input (**"yes"** or **"no"**). Other entries trigger NoMatch events.

2. This element then directs the callflow down the "yes" or "no" exit state based on the caller's input. To use DTMF entries other than 1 for yes and 2 for no, use the 2_Option_Menu instead.

3. There is an optional Setting that allows the caller to say **"replay"** or press **DTMF 3** to replay the **Initial** audio group to the caller.

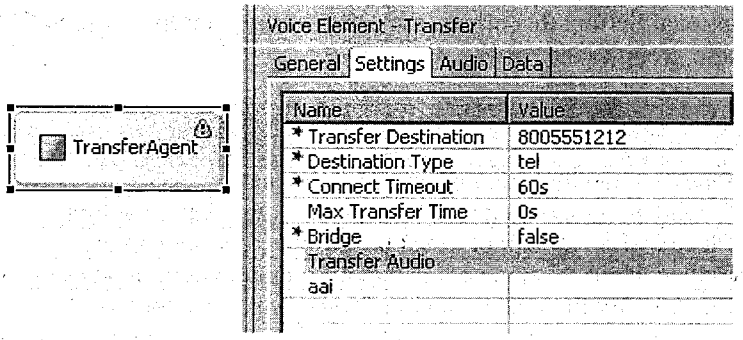
| **Audio** | |
|---|---|
| **Audio Group Name** | |
| **Initial** | Played when the voice element first begins. |
| **NoMatch** | Played when a nomatch event occurs during input capture. |
| **NoInput** | Played when a noinput event occurs during input capture. |
| **Help** | Played when a help event is triggered during input capture by caller saying 'help' |
| **End > Add Yes** | Played after the caller confirms (says "yes" or presses 1). |
| **Settings** | |
| **Input Mode** | The type of entry allowed for input. Possible values are: voice \| dtmf \| both. |
| **Noinput Timeout** | The maximum time allowed (in seconds) for input before a **NoInput** event is thrown. |
| **Max NoInput Count** | The maximum number of noinput events allowed during collection. |
| **Max NoMatch Count** | The maximum number of nomatch events allowed during collection. |
| **Confidence Level** | The confidence level threshold to use during collection. |
| **Replay** | **Indicates whether the caller can say "replay" or press DTMF 3 to hear the Initial Prompt again.** |
| **Element Data** | |
| **value** | The value selected by the caller (can be "**yes**" or "**no**") |
| **value_confidence** | This is the confidence value of the captured utterance. |
| | |
| **Exit States** | |
| **yes** | The caller said "yes" or pressed DTMF 1 |
| **no** | The caller said "no" or pressed DTMF 2 |
| **max_nomatch** | Max number of nomatch events has occurred. |
| **max_noinput** | Max number of noinput events has occurred. |

## Transfer Element (Folder: Elements/Call Control/)

1. Often if a caller follows the max_nomatch or max_noinput paths, they need to be transferred to an agent.

2. In a **CVP Comprehensive environment**, use a **CVP_Subdialog_Return** element to return call control to ICM to transfer the call. Use the Caller Input, and FromExtVXML settings to pass information to ICM to transfer to the correct skill group and pass relevant screen pop information.

3. In a **CVP Standalone** deployment, use the **Transfer Element** to transfer the call to an external PBX, ACD, or PSTN. You can not pass data unless the destination application has access to AAI (Application-to-Application Information).




4. The transfer can be a **bridge transfer** or a **blind transfer.** For more complex transfers, see the ***CVP Config and Admin Guide.***

5. For a **bridge transfer**, the voice browser makes an outbound call while maintaining the original call and acts as a bridge between the two calls. Oonce the third party disconnects, the original call continues with the IVR.

**NOTE** – With bridge transfer, ensure the **Project Properties > Session Timeout** is large enough to encompass the amount of time allowed for the bridge referral.

6. For a **blind transfer**, the voice browser makes an outbound call to the callee and ends the VXML Gateway and VXML Server connection regardless of the outcome.

7. The Transfer element defines **exit states** for the different ways bridge transfers can end such as the person being called hung up, there was no answer, there was a busy signal, or some other phone-related error occurred.

8. The number to transfer to can be any phone number allowed by the voice browser telephony provider. Check the browser documentation for specific requirements and restrictions for call transfer.

## Transfer Element Settings

1. **Transfer Destination** The phone number to transfer to. It may contain nonnumerical characters to allow support for phone extensions and you may use a variable to specify the phone number.

2. **Destination Type** The type of transfer destination to which the voice element is to connect. Possible values are: tel | sip.

3. **Connect Timeout** The number of seconds the voice element is allowed to wait for an answer, after which it will exit with the "noanswer" exit state. Note that this setting only applies to the bridge mode.

4. **Max Transfer Time** The maximum duration (in seconds) the transfer is allowed to last. By default, it is an infinite amount of time, so the setting should be configured only if a limit is required. Note that max_transfer_time only applies to the bridge mode.

5. **Bridge** This setting determines what to do after the call is connected. When set to true, the call continues in the Studio application after the third party disconnects their leg of the call. When set to false the Studio application terminates immediately.

6. **Transfer Audio** The URI of an audio file to be played while connecting the call. For example, you can play a ring tone as prerecorded audio to the caller.

7. **Application-to-application information** A string containing Application-to-Application Information data to be sent to an application on the far-end.

## Element Data

**result** The result of the transfer, such as:

- near_end_disconnect (incoming caller hung up)
- far_end_disconnect (callee hung up)
- network_disconnect, busy (transfer failed because the called number was busy)
- network_busy
- noanswer (transfer failed because there was no answer and the connect timeout expired)

## Audio Groups

1. **Initial** Played to the caller before the transfer.
2. **Busy** Played when there is a busy signal, right before the voice element exits with the "busy" exit state.
3. **No Answer** Played when there is no answer, right before the voice element exits with the "noanswer" exit state.
4. **Phone Error** Played when there is some kind of phone-related error, right before the voice element exits with the "phone_error" exit state.
5. **Done** Played for a bridge transfer when the call transfer completes when the initial caller is still on the line.

## Reference: Changing an X-option Menu to (X+1)-option Menu

If you need to change the number of exit states for an N_Option_Menu element, you may either create a new element with more exit states and copy/paste all the settings and audio; or use the following method to modify the type of element Studio thinks this element is.

A. NOTE - This method has you modify the XML file that tells Studio the type of element being used. **It is not to be executed without taking the following precautions!!!**

B. ***Back up the Studio application*** **YOU'VE BEEN WARNED!**
(In the Navigator, right-click copy/paste)

C. Note the **name of the element** and the **page name** on which it resides.

**D. Close the application out of the workspace.**

E. Open this file in Notepad:
C:\Cisco\CallStudio\eclipse\workspace\ <projectname>\callflow\pages\<**pagename**>**\.page**

F. Find the name of the menu element and change MFoundation2OptionMenu to the desired number of options (2-10 are valid): **MFoundation3OptionMenu.**

```
<element height="46" id="Element@31077784:1224106268968" name="mMainMenu"
src="com.audium.server.voiceElement.menu.MFoundation2OptionMenu"
type="Voice Element" width="144" x="142" y="234">
<exit_states>
```

G. Save and ***QUIT*** Notepad, **YOU'VE BEEN WARNED!**

H. In Studio, double-click the **app.callflow** to open the application in Studio. The Menu has a new exit state and the Settings tab has another option to configure.

**Chapter 3 Review Questions**

1. How do you start the Call Studio Debugger to test the application without VxmlServer? How do you terminate the Debugger? How do you go back to the Studio view after running the Debugger?

2. In which directory are the Debugger's log files stored?

3. What is the name of the audio group that plays when the caller enters an invalid selection at a Menu element?

4. What is the name of the audio group that plays when the caller requests 'help'

5. Suppose you want the caller to experience at most 3 problems (combination of NoMatch and NoInput events) before being transferred to an agent, how would you accomplish this?

6. When does the Help audio group play?

7. At a menu element, where is the caller's input stored?

8. What steps must be taken to collect # as a menu selection from the caller? Why?

9. Where do you configure the amount of time the gateway waits for the caller to start entering something before throwing a NoInput event?

## Chapter 3 Programming Lab

Element names in **bold** will be referred to in later labs. Please use the recommended names.

1. Create a new CVP Project, named **MyApp**.

2. The instructor will tell you which Gateway Adapter to use

3. Use a CVP Subdialog Start element.

4. Create an Audio element (named **pWelcome**) and speak the TTS prompt: *Welcome to CVP Insurance.*

5. Add a 4-Option-Menu (named **mInsType)** to collect the insurance type from the caller.

A. Audio Tab:

- **Initial Prompt**: *Which insurance account are you calling about? Auto insurance, medical insurance or dental insurance. You may press pound to speak to an agent.*
- **NoMatch1:** *I didn't understand. Say auto insurance or press 1. Say medical insurance or press 2. Or say dental insurance or press 3.*
- **NoMatch3:** *I just can't understand.*
- **End >> Done:** *You selected {Data.Element.mInsType.value}* (Use the {} button to enter the variable reference)

B. Settings Tab:

1) NoInput Timeout: 2s
2) Confidence Level: 0
3) Option 1
   DTMF: **1**
   Voice: **auto**
   Voice: **auto insurance**
   Value: **auto insurance**
4) Option 2
   DTMF: **2**
   Voice: **medical**
   Voice: **medical insurance**
   Value: **medical insurance**
5) Option 3
   DTMF: **3**
   Voice: **dental**
   Voice: **dental insurance**
   Value: **dental insurance**
6) Option 4
   DTMF: #
   Voice: **agent**
   Value: **an insurance agent**

7) **VoiceXML Property - at the bottom of the Settings tab, enter the following:**
   - Enter the VoiceXML Property name: **termchar**
   - Enter the Value: **(enter one blank space in the value field)**

6. Create two more pages.
   - Name page2 and its Entry Connector **GetPayment**
   - Name page3 and its Entry Connector **CSR**

7. On page 1, bring in 2 page connectors. Configure one for **GetPayment** and the other for **CSR**.

8. Connect the **mInsType option1, option2, and option3** exit states to the page connector that goes to the **GetPayment** page. Bend the exit states so they are visible.

9. Connect the **mInsType option4, max_nomatch, max_noinput** exit states to the Page Connector that connects to the **CSR** page. Bend the exit states so they are visible.

**Create the following on the GetPayment page:**

10. Create an Audio element (named **pBalance**) to say:
    - audio item1: ***Your {Data.Element.mInsType.value} balance is $1000***

11. Create a Yes_No_Menu (named **ynMakePayment**) to ask the caller if they would like to make a payment.
    A. Audio
       - **Initial Prompt**: *Are you calling to make a payment toward your insurance bill? If so, please say yes or press 1. Otherwise, say no or press 2. To hear these instructions again, say re-play or press 3.*
       - **NoMatch1:** *If you would like to make a payment, press 1. If not, press 2.*
       - **NoMatch3:** *I just can't understand.*

    B. Settings
       - Confidence Level: 0
       - Replay: True

12. **If the caller says Yes**

    A. Connect the **ynMakePayment → yes** exit state to a new Audio element (named **pHoldPayments**). The **pHoldPayments** element should say:
    *We will now pass your payment information to the next available agent, please hold..*

    B. Connect the **pHoldPayments** element to a new **CVP Subdialog Return** element. Set the following:
       - Caller Input: **payments**
       - External VXML 0: **{Data.Element.mInsType.value}**

13. **If the caller says No**

    A. Connect the **ynMakePayment → no** exit state to a new Audio element **pGoodbye**, that says to the caller ***Thank you for calling, goodbye.***

    B. Use a **CVP Subdialog Return** element and set Caller Input: **done**

14. Connect the **ynMakePayment →max_nomatch** and **max_noinput** exit states to a new Page Connector that goes to the **CSR** page.

**The following code will be created on the CSR page:**

15. Create an Audio element (named **pCSR**) that tells the caller to *please hold for a customer service representative.*

16. Connect this to a new **CVP Subdialog Return** element where you configure Caller Input setting with the word **transfer**.

**Save and deploy the application from Studio.**

**Run the updateApp.bat script.**

**Call in and test the application.**

**When done testing, rename this application 'Insurance'**

## Challenge (only if you are done early)

Convert the 4 option menu into a 5 option menu:

1. In Studio, right click MyApp in the Navigator and select Copy. Then right-click and select paste. This makes a back up of your application.

2. Close the MyApp application from the Studio Workspace (click the x at the top of the callflow editor window).

3. Use the method in this chapter to convert the 4 option menu (mInsType) into a 5 option menu.

**CAUTION!** 4. **REMEMBER** to **QUIT** out of Notepad after modifying the .page file. Otherwise you will corrupt the Studio application and make it unusable later in the class.

5. The 5th option will be for **'flood insurance'** (or DTMF 4)

6. Open the application in Studio and ensure that it has 4 exit states, and if necessary modify the audio prompts and the configuration. Test the application.

[End of Chapter 3]

# Chapter 4

# Collecting Custom Input

- Collecting Caller Input: Digits, Number, Currency, Date, Time, Phone
- VoiceXML Properties
- Say it Smart Plugins
- Elements that Collect & Confirm

## Voice Elements That Collect Input

1. Many of the **Voice** elements in Call Studio are used to collect information from the caller. Each Voice element creates one or more VXML pages executed by the voice browser.

2. Some of the elements: Digits, Number, Phone, Currency, Date, Time and YesNo_Menu use **"builtin grammars"** provided with the VoiceXML Gateway and the ASR system. These require caller input to be in a specific format determined by the VoiceXML Specification. The builtin grammars include: digits, number, currency, date, time, phone, boolean.

3. The Voice elements store the caller's input into Element data called **'value'** that 'belongs to' the Element that collected it. The format of the returned value is determined by the W3C VoiceXML Specification.

## The Digits Element (Folder: Elements/Number Capture/)

1. The **Digits** element is an example of a **Voice** element that collects input from the caller and stores the value into an element data variable called '**value**'. It allows the caller to press or say **one digit at a time** as input.

2. This element accepts the digits 0-9

| Audio Groups | |
|---|---|
| **Initial** | Initial prompt spoken to the caller. System then waits for caller input. |
| **NoMatch1,2,3** | Spoken if caller input doesn't match a menu option or if confidence is below the cutoff value. System then waits for caller input. |
| **NoInput1,2,3** | Spoken to the caller if the NoInput timer expires. System then waits for input. |
| **Help1,2,3** | Spoken to the caller if caller says 'help' or if the event named 'help' occurs. System then waits for caller input. |
| **End >> Done** | Spoken to the caller only after valid input is collected. |

| Settings | |
|---|---|
| ***Input Mode** | How caller may enter input: voice, dtmf, both |
| ***Noinput Timeout** | The maximum time for caller to start entering something before a **NoInput** event is thrown. Use a time unit (s for seconds, ms for milliseconds). E.g., 3500ms or 5s |
| ***Max NoInput Count** | The maximum number of noinput events allowed during collection. |
| ***Max NoMatch Count** | The maximum number of nomatch events allowed during collection. |
| ***Confidence Level** | The confidence threshold during collection. ASR results with confidence below this cause a **nomatch** event and no data is returned to the gateway. (DTMF entries are assigned confidence 1.0). Default 0.5 |
| ***Disable Hotlinks** | Whether to disable all hotlinks (all other active grammars besides what this element specifically collects) |
| ***Min Digits** | The minimum number of digits to accept from the caller. Entering fewer digits than this forces a NoMatch event to occur. |
| ***Max Digits** | The maximum number of digits the gateway should 'listen for' and accept. . Extra touchtones are stored by the gateway for type-ahead (key-ahead) and will barge in on succeeding prompts. |
| ***Disable Hotlinks** | Whether or not to disable all global and local Hotlinks (default: false) |
| **Secure Logging** | Whether or not to disable logging of potentially sensitive data (default: false).<br>If **'true'** the data and interaction will log as the name of the variable followed by **_secureLogging** and with value ***** and the information will appear in the logs or reporting server. |
| 03/19/2008 12:38:35.826,getAcct,interaction,utterance_secureLogging,*****<br>03/19/2008 12:38:35.830,getAcct,interaction,interpretation_secureLogging,*****<br>03/19/2008 12:38:35.843,getAcct,data,value_secureLogging,*****<br>03/19/2008 12:38:35.843,getAcct,data,confidence,1<br>03/19/2008 12:38:35.843,getAcct,data,nbestUtterance1_secureLogging,*****<br>03/19/2008 12:38:35.843,getAcct,data,nbestInterpretation1_secureLogging,***** | |
| ***Maxnbest** | (ASR only) – Specify the maximum number of recognition results to be returned by the speech recognizer. Usually this is left at 1. If set to a value greater than 1, the element data **nbestLength** contains the number of results returned. The results are returned into element data named: **nbestInterpretation1,** |

| | |
|---|---|
| | **nbestInterpretation2, etc**. The corresponding confidence values are in element data named **nbestConfidence1**, **nbestConfidence2**, etc. |

| Exit States | |
|---|---|
| **done** | The path followed when caller input is successfully captured |
| **max_nomatch** | Max number of nomatch events has occurred. |
| **max_noinput** | Max number of noinput events has occurred. |

| Element Data created by this element upon successful collection of caller input | |
|---|---|
| **value** | The digits representing the caller's entry |
| **value_confidence** | Confidence value assigned by the ASR (0.0-1.0) DTMF entry is always 1.0 |
| When working with multiple results from the recognizer, the following Element data variables will be repeated, with the '1' replaced by a 2, 3, . . .<br><br>**Most of these Element data variables are NOT listed in the Substitution Tag builder.** To access the other variable you must **manually type them** into the Substitution tag builder. | |
| **nbestLength** | The number of values returned by the recognizer when working with maxnbest > 1 |
| **nbestInterpretation1** | Same as 'value' variable. Contains the interpretation returned from the grammar based upon the caller's input. |
| **nbestConfidence1** | Same as 'value_confidence'. |
| **nbestUtterance1** | Contains the utterance, the words spoken by the caller, as it matched the grammar. |
| **nbestInputmode1** | Contains the mode of caller input, either 'dtmf' or 'voice'. |

## Programming Notes

1. DTMF entry: once the system collects the **maximum number of DTMF digits** from the caller, it stops listening for input. So if the caller enters max digits followed by the **terminating character (#),** the # remains in the DTMF-input buffer and causes the gateway to abort successive audio and causes a nomatch event at the next caller input element.

2. To prevent this, you can do any one of the following:

   A. Within the Settings of this element, set the VoiceXML Property '**termtimeout**' to a time unit <= to your **interdigittimeout** (e.g., **3s**) so the gateway waits for the terminating character (#) once the MaxDigits has been collected.

   If the extra DTMF tone is the **termchar** (#), it will be stripped from the input. If the extra DTMF entry is a different touchtone, a nomatch event occurs.

   B. Or you can set **MaxDigits** to be larger than the maximum number of digits you are actually collecting from the caller.

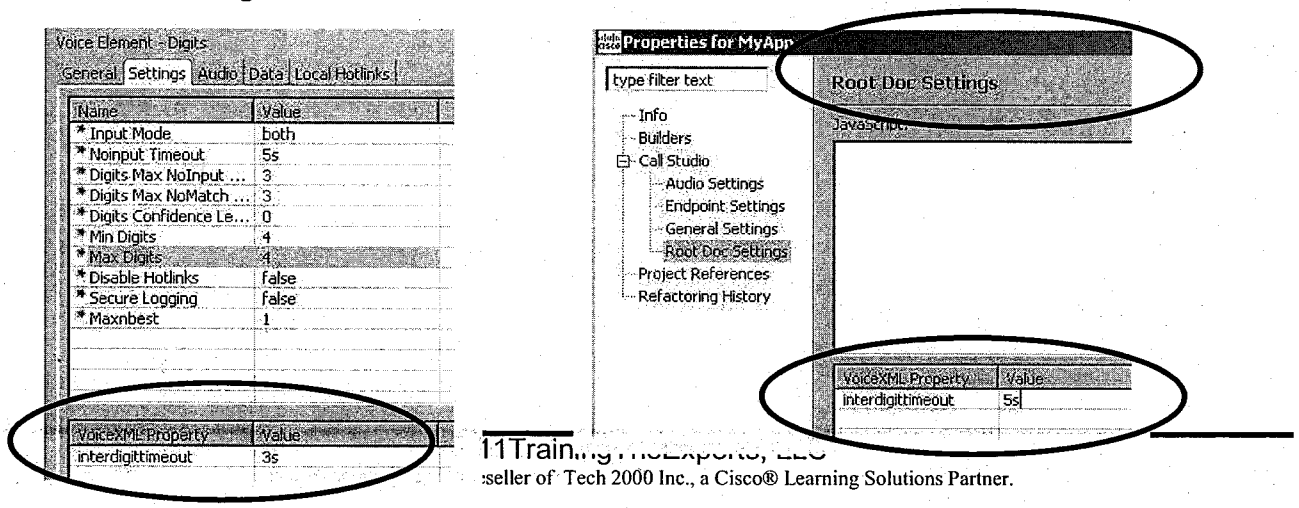
   C. Or you can clear the DTMF input buffer by using another Digits element to speak silence_.1_sec.wav with **Bargein Disabled**, set Noinput Timeout to 10ms, set MaxNomatchCount and MaxNoinputCount to 1, set MinDigits and MaxDigits to 1, and connect all exit states to the next element in the call flow.

## VoiceXML Collection Properties

1. VoiceXML properties effect the VoiceXML Gateway and include prompt and collect timers, fetch timeout timers, caching properties, etc.

2. You can set VoiceXML properties and values at the bottom of the Settings tab of any Voice Element, where it only applies to this one element.

3. To configure a property for the entire application, use the **Project Properties / Root Doc Settings.**



4. If you set a property but omit a value, an 'error.badfetch' event occurs at runtime.

| **DTMF Properties** | All time values (except audiomaxage) MUST end in '**s**' for seconds or '**ms**' for milliseconds. |
|---|---|
| interdigittimeout | The inter-digit timeout value to use when recognizing DTMF input. This timer is used when the caller has not entered enough touchtones to satisfy the application's requirements. **VXML Server Default: 10s** |
| termtimeout | This is used when the VoiceXML Gateway has collected the maximum number of digits requested, so that it waits for a possible '#' terminating character from the caller.<br><br>If the # is entered, it will be stripped from the input. If some other digit is entered, a NoMatch event occurs. If this timer expires without any more digits being entered, the application continues successfully.<br><br>**Default: 0s (disabled).** Normally this setting is disabled for 1-digit input (e.g., Menu entries) |
| termchar | The terminating DTMF character for DTMF input recognition. **Default:** # |
| **ASR Properties** | |
| incompletetimeout | **Nuance OSR**: this timer indicates the caller is done speaking. It should be set very short for one word responses ('yes', 'no'), but made longer when multiple word responses are possible. Analogous to the DTMF interdigittimeout. Default: platform-dependent |

## Say it Smart - Speaking Formatted Data

1. In order to speak the value of a variable in a specific format, the variable may have to be parsed and converted into a set of audio files (and backup TTS text) to play.

**Example, '1234'** spoken as DIGITS must be converted into:

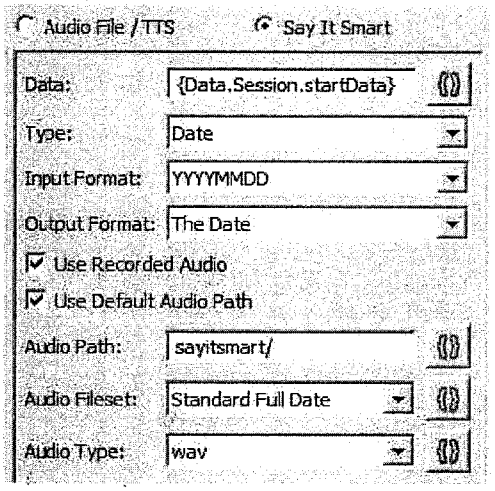
```
<prompt bargein="true">
    <audio src="http://10.1.78.12/audio/sys/1.wav">one</audio>
    <audio src="http://10.1.78.12/audio/sys/2.wav">two</audio>
    <audio src="http://10.1.78.12/audio/sys/3.wav">three</audio>
    <audio src="http://10.1.78.12/audio/sys/4.wav">four</audio>
</prompt>
```

**Example, '20090101'** spoken as a date must be converted into:

```
<prompt bargein="true">
    <audio src="http://10.1.78.12/audio/sys/january.wav">january</audio>
    <audio src="http://10.1.78.12/audio/sys/2nd.wav">    second</audio>
    <audio src="http://10.1.78.12/audio/sys/2.wav">  two</audio>
    <audio src="http://10.1.78.12/audio/sys/thousand.wav">thousand</audio>
    <audio src="http://10.1.78.12/audio/sys/9.wav"> nine</audio>
</prompt>
```

2. **Say It Smart** is a Cisco VXML Server technology that handles the conversion of data into a set of audio files (and/or Text to Speech prompts) to render the data in a manner understandable by a caller.

3. Each **Say It Smart** type lists the audio files required to fully render all the formatted data it can handle. The operator need only record these files according to the guidelines specified in the ***Cisco Say it Smart Specifications reference manual*** and Say It Smart does the rest.



4. **Settings**:

A) **Type**. A Say It Smart plugin is associated with a single type that defines what kind of data can be handled by the plugin. Number, Date, and Currency are examples of Say it Smart types.

B) **Input Format**. Defines how the plugin parses the variable. For example, a date may appear in MMDDYY or YYYYMMDD.

C) **Output Format**. Defines how to express the spoken output. Output formats are dependent on input formats, once an input format is selected, the available output formats may change. For example, for a time in this input format (HHMM), you may specify either the output format that reads 12:00 as "noon" or as "12:00 pm".

D) **Use Recorded Audio** – select this for the Say it Smart to generate a list of audio files to speak the specified data

- **Use Default Audio Path** – whether to prepend file names with the default audio path configured in Project / Properties / Audio Settings

- **Audio Path** –prepended to the audio file name. If used along with the Default Audio Path, then the order is:
  **DefaultAudioPath + Audio Path + file name + AudioType file extension**

- **Audio Fileset** - The filesets are the audio files required to render a particular output format. The most common use of different filesets is to render the data so it sounds better to the caller. This **might** be done by performing less concatenation of prompts. Another use for filesets would be to provide a different gender or playback speed

- **Audio Type** – wav, vox, au. This will be appended as a file extension to the name of the audio files.

5. **Debugging Tips:**

A. Say it Smart plugins are **not fault tolerant** and often require you to preprocess the data to ensure the **variable is not empty** and has **no invalid characters** for the selected Say it Smart type. It is best to test all possible situations to know which Say it Smarts can accept partial data.

B. The media files installed with CVP include audio files named correctly for MicroApps, but named **incorrectly** for the Date Say it Smart. You must copy/paste the files **1ord, 2ord, 3ord,** (etc) and name them **1st, 2nd, 3rd, 4th** as specified in ***Say it Smart Specifications.pdf***

C. The **Credit Card** Say it Smart causes an error unless you have a TTS synthesizer to play the pauses it generates between sets of digits.

6. The list of audio file names required for each Say it Smart plugin is available in the reference manual ***Say it Smart Specifications for Cisco CVP.***

## Formatted Input Elements (Currency, Number, Phone, Time, Date)

The **Currency, Number, Phone, Time, and Date** voice elements capture formatted input and return it in a predetermined format:

1. **Currency Element** (Folder: Elements/Commerce/) Use this to collect currency amounts from the caller. For DTMF whole dollar entries have callers press the # sign to terminate their entry.

   a) **ASR:** Callers say currency amounts followed by the word dollars and/or cents. The spoken currency unit (Dollars, Pounds, Euros, etc) will be specified by the ASR company and the language/locale used. Consult the ASR documentation.
   b) **DTMF: DDDD[*CC]** where DDDD represent some number of touchtones representing the dollars. The optional * key represents a decimal point. And the optional CC represents 2 touchtones representing cents. **For whole dollar amounts, callers should terminate their entry with the # sign.**
   c) **Returned Value:** Element data '**value**' containing a decimal value with up to 2 padded zeroes; does not include a currency or dollar sign ($).

2. **Number Element** (Folder: Elements/Number Capture/) –Collect a **number** from the caller. E.g., collect caller's age, number of months to pay, their temperature.

   a) **ASR**: The spoken number can be negative or positive and can contain a decimal point. Using speech input, the number may be spoken naturally.
   b) **DTMF**: The number must be positive and the optional decimal point is entered by pressing the * key. **Callers should terminate their entry with the # sign.**
   c) **Returned Value**: Whole or decimal number is returned.

3. **Time** (Folder: Elements/Date&Time/): Use this to collect a time from the caller. Note that Time is returned as a 5-character string (HHMMx).

   a) **ASR**: Callers say the time naturally followed, optionally, by A.M., P.M., midnight, noon, morning, evening.
   b) **DTMF**: Time is entered as 4-digits (24 hour format) HHMM. The system does **NOT** wait for the # key.
   c) **Returned Value**: Time is returned as a 5-character string HHMMx where x is either 'a' (AM), 'p' (PM), 'h' (24-hour format), or '?' (if unspecified).

| ASR Caller says: | value returned | DTMF Entry | value variable |
|---|---|---|---|
| ***Two thirty*** | **0230?** | **0230** | **0230?** |
| ***Two thirty AM*** | **0230a** | **1430** | **1430h** |
| ***Two thirty PM*** | **0230p** | | |
| ***Fourteen Thirty*** | **1430h** | | |

4. **Date** (Folder: Elements/Date&Time/): This element is rarely used if you are collecting DTMF entry as it requires the caller to enter an 8-digit date in YYYYMMDD format.

   a) **ASR**: Callers may say the full day (month, day and year), a portion of the date (month, day or year), or today, tomorrow, yesterday.
   b) **DTMF**: Date is entered as 8-digits YYYYMMDD format. The system does **NOT** wait for the # key.
   c) **Returned Value**: Date is returned as an 8-digit string YYYYMMDD format. Omitted data appears as '?'

| Caller Says | value variable | DTMF entry |
|---|---|---|
| *January sixth 2010* | 20100106 | 20100106 (YYYYMMDD) |
| *January sixth* | ????0106 | |
| *January* | ????01?? | |
| *Today* | 0 | |
| *Tomorrow* | +1 | |
| *Yesterday* | -1 | |

5. **Phone** (Folder: Elements/Number Capture/): :

   a) **ASR**: Callers may say the phone number (including an optional extension) in natural language.
   b) **DTMF**: The caller can enter a phone number. An optional extension can be indicated by pressing the * key followed by the extension.
   c) **Returned Value**: The phone number entered with an optional character "x" followed by the extension.

6. The **Settings, Audio, Element Data and Exit States** are the standard for voice element configuration.

| Audio | |
|---|---|
| **Audio Group Name** | **Notes** |
| **Initial** | Played when the voice element first begins. |
| **NoMatch** | Played when a nomatch event occurs during input capture. The nomatch event count corresponds to the audio group count. If not specified, the initial audio group will be played in the event of a nomatch. |
| **NoInput** | Played when a noinput event occurs during input capture. The noinput event count corresponds to the audio group count. If not specified, the initial audio group will be played in the event of a noinput. |
| **End** | |
| **Done** | Played after the input is successfully collected. |

| Settings | |
|---|---|
| **Input Mode** | The type of entry allowed for input. Possible values are: voice \| dtmf \| both. |

| | |
|---|---|
| **Noinput Timeout** | The maximum time allowed (in seconds) for silence or no keypress before a **NoInput** event is thrown. |
| **Max NoInput Count** | The maximum number of noinput events allowed during collection. 0 = infinite noinputs allowed. |
| **Max NoMatch Count** | The maximum number of nomatch events allowed during collection. 0 = infinite nomatches allowed. |
| **Confidence Level** | The confidence level threshold to use during collection to determine when to throw the 'nomatch' event. Float between 0.0 – 1.0. |
| **Disable Hotlinks** | Whether or not to disable all global and local Hotlinks (default: false) |
| **Secure Logging** | Whether or not to enable logging of potentially sensitive data (default: false). |
| **Maxnbest** | Specifies the maximum number of recognition results returned by the speech recognizer |

| Element Data | |
|---|---|
| **value** | The input stored in the VXML Specified format |
| **value_confidence** | This is the confidence value of the captured utterance., between 0.0 and 1.0 (1.0 for DTMF) |
| **nbestInputmode1,...** | The input mode 'voice' or 'dtmf' |
| **nbestUtterance1,,...** | The utterance from the caller as it matched the grammar |
| **bestConfidence1,...** | The confidence value returned by the recognizer. Float between 0.0 – 1.0 (DTMF is 1.0) |
| **nbestInterpretation1 ,...** | The interpretation of the caller's input as returned by the recognizer |
| **nbestLength** | The number of results returned from the recognizer when maxNbest is set to a number greater than 1. |

| Exit States | |
|---|---|
| **max_nomatch** | The maximum number of nomatch events has occurred. If the nomatch max count is 0, this exit state will never occur. |
| **max_noinput** | The maximum number of noinput events has occurred. If the noinput max count is 0, this exit state will never occur. |
| **done** | The time capture was completed |

## Debugging Applications

1. Say it Smart accounts for a large percentage of the errors that occur!

2. If an error occurs when calling into an application consult the application level logs:
   a. **First consult the Activity Log** to determine the bad element.
   b. **Next, consult the Error Log** for detailed information. The error log often contains 3 stack traces. **Scroll up from the bottom until you find the Session ID** - here you'll find the initial cause of the problem.

**NOTE** - Logs are **cached** into memory **until the call ends** or the memory cache becomes full.

**Example: Activity Log**

- 11:34:23,get_paydate,element,enter,
- 11:34:23,get_paydate,interaction,audio_group,collect_initial_audio_group
- 11:34:30,get_paydate,interaction,utterance,eighth
- 11:34:30,get_paydate,interaction,inputmode,voice
- 11:34:30,get_paydate,interaction,interpretation,??????08
- 11:34:30,get_paydate,interaction,confidence,0.25
- 11:34:30,get_paydate,data,value,??????08
- 11:34:30,get_paydate,data,value_confidence,0.25
- 11:34:30,get_paydate,element,exit,done
- 11:34:30,yn_paydate,element,enter,
- **11:34:30,yn_paydate,element,error,**

**Sample Error Log (3 different examples of Say it Smart errors)**

a. An element encountered an exception of type com.audium.server.xml. ElementConfigException with the message: **SayItSmart Error - Digits**: The **data passed as input "123*"** must conform to one of the following formats: " dddddd ".

Variable contained invalid data

b. An element encountered an exception of type com.audium.server.xml. ElementConfigException with the message: **SayItSmart Error – Credit Card**: The **data passed as input "20080908"** must conform to one of the following formats: "dddd-dddd-dddd-dddd" "ddddd-ddddd-ddddd" or "dddd-dddd-ddddd"

You selected the wrong Say it Smart Type

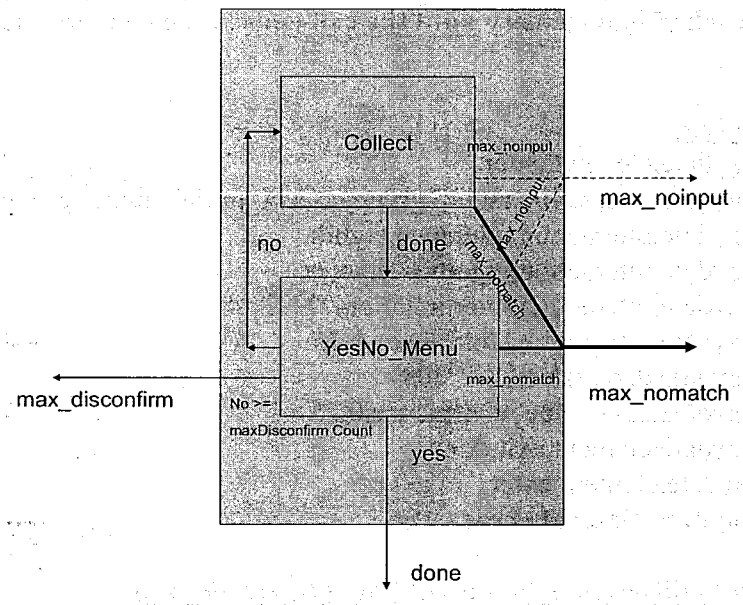
c. An element encountered an exception of type com.audium.server.xml. ElementConfigException with the message: **SayItSmart Error** – Digit by Digit: The **data passed as input " "** must conform to one of the following formats: "dddddd"

You selected the wrong (or an empty) variable

## Voice Elements that Collect and Confirm

1. The Digits, Number, Time, Currency, and Phone elements have a corresponding voice element that **collects the input and then confirms it** with the caller before continuing.

2. Internally, the element executes 2 elements – the Collection stage collects the input (Digits, Number, Time, etc) and the Confirmation stage is really a YesNo_Menu. The CollectWithConfirm elements are a Studio short cut but are no more efficient than coding the elements yourself.




3. A **Disconfirmation** means the caller says 'No' or presses 2 at the YesNo_Menu stage.

4. You can configure the setting **Max Disconfirmed Count** for the number of times the caller may indicate '**no**' and be reprompted for input. Once the maximum number of disconfirmations is reached, the callflow exits down the **max_disconfirmed** exit state.

5. The application exits down the '**done**' exit state only if the caller confirms the input.

6. **WARNING**: Do NOT use the **DateWithConfirm** element as you **must** parse the collected date to ensure it is complete (no '?' and the correct number of digits) before speaking it using Say it Smart..

7. **WARNING** – When using the DigitsWithConfirm element, if you enable the **termtimeout**, then the caller must press # to terminate DTMF entry during the collection stage **AND during Confirmation stage**. This will seem unnatural for most callers.

| Configuring Audio | |
|---|---|
| **Collection Stage** | |
| **Audio Group Name** | **Notes** |
| **Collect Initial** | Played when the voice element first begins. |
| **Collect NoMatch** | Played when a nomatch event occurs during data input capture. The nomatch event count corresponds to the audio group count. If not specified, the initial audio group will be played in the event of a nomatch. |
| **Collect NoInput** | Played when a noinput event occurs during data input capture. The noinput event count corresponds to the audio group count. If not specified, the initial audio group will be played in the event of a noinput. |
| **Collect Help** | Played when a help event is triggered during data input capture. The help event count corresponds to the audio group count. By default, the help event can be triggered by the voice input "help". Additionally, other voice and DTMF inputs can be used to trigger the help event by configuring a hotlink to throw a help event. If the audio group is not specified, a help event will trigger the nomatch audio group to play (or the initial audio group if nomatch audio group is not specified). |
| **Confirmation Stage** | |
| **Confirm Initial** | Played when confirmation first begins. |
| **Confirm NoMatch** | Played when a nomatch event occurs during confirmation. The nomatch event count corresponds to the audio group count. If not specified, the confirm initial audio group will be played in the event of a nomatch. |
| **Confirm NoInput** | Played when a noinput event occurs during confirmation. The noinput event count corresponds to the audio group count. If not specified, the confirm initial audio group will be played in the event of a noinput. |
| **Confirm Help** | Played when a help event is triggered during confirmation. The help event count corresponds to the audio group count. By default, the help event can be triggered by the voice input "help". Additionally, other voice and DTMF inputs can be used to trigger the help event by configuring a hotlink to throw a help event. If the audio group is not specified, a help event will trigger the confirm nomatch audio group to play (or the confirm initial audio group if confirm nomatch audio group is not specified). |
| **Disconfirmed** | Played after the caller disconfirms a captured data entry. Upon reaching the **MaxDisconfirmedCount**, the prompt should indicate that the caller is exiting down the **max_disconfirmed** exit state. |
| **End** | |
| **Yes** | Played after the caller chooses the "yes" option. If not specified, no audio will be played. |
| | |
| **Settings** | |
| **Input Mode** | The type of entry allowed for input (during data capture and confirmation). Possible values are: voice \| dtmf \| both. |
| **Noinput Timeout** | The maximum time allowed (in seconds) for silence or no keypress before a **NoInput** event is |

| | |
|---|---|
| | thrown. |
| **Collect Max NoInput Count** | The maximum number of noinput events allowed during Data collection stage. |
| **Collection Max NoMatch Count** | The maximum number of nomatch events allowed during Data collection stage. |
| **Confirm Max NoIput Count** | The maximum number of noinput events allowed during the Confirmation stage. |
| **Confirm Max NoMatch Count** | The maximum number of nomatch events allowed during the confirmation stage. |
| **Max Disconfirmed Count** | The maximum number of times a caller is allowed to disconfirm (say "no" or press "2") a captured input. |
| **Data Confidence Level** | The confidence level threshold to use during data capture. |
| **Confirm Confidence Level** | The confidence level threshold to use during confirmation. |
| **Min Digit (Digits element only)** | Minimum number of digits allowed. Anything less than this results in a **NoMatch** event |
| **Max Digit(Digits element only)** | Maximum number of digits allowed. |
| **Element Data** | |
| **value** | The digit string captured. |
| **value_confidence** | This is the confidence value of the captured digit string utterance. |
| **confirm_confidence** | This is the confidence value of the captured confirm utterance. |
| nbestInputmode1, 2,3,… | The input mode 'voice' or 'dtmf' |
| nbestUtterance1, 2,3,… | The utterance from the caller as it matched the grammar |
| bestConfidence1, 2,3,… | The confidence value returned by the recognizer. Float between 0.0 – 1.0 (DTMF is 1.0) |
| nbestInterpretation1, 2,3,… | The interpretation of the caller's input as returned by the recognizer |
| nbestLength | The number of results returned from the recognizer when maxNbest is set to a number greater than 1. |
| **Exit States** | |
| **max_nomatch** | Max number of nomatch events has occurred. |
| **max_noinput** | The maximum number of noinput events has occurred. |
| **max_disconfirmed** | The maximum number of disconfirmation has occurred. |
| **done** | The digit string captured was confirmed. |

## Voice Element Reference

Most of the following Voice elements collect input from the caller using "builtin grammars". This table describes the valid caller input for DTMF and voice inputs and the format of data returned into the Studio 'value' variable.

| Name | Description | Return value | Returned Format |
|---|---|---|---|
| Commerce/ **Currency** | Captures a currency amount in dollars and cents.<br>**DTMF:** use * for the decimal point.<br>**Voice**: dollars and optional cents. | value<br>value_confidence<br>nbestInputmode1<br>nbestUtterance1<br>nbestConfidence1<br>nbestInterpretation 1<br>nbestLength | **DDD.CC** |
| Date&Time **/Date** | Collects a date using the built-in date grammar.<br>**DTMF: YYYYMMDD or YYMMDD**<br>**Voice:** Any spoken date that includes the month and day. | | **YYYYMMDD** |
| Date&Time **/Time** | Collects a time using the built-in time grammar.<br>**DTMF: HHMM** (HH is in 24-hour format)<br>**Voice:** Any spoken time | | **HHMMX** |
| Form/ **Form** | Capture custom input. Specify external grammars (file or URI), inline grammars (keywords), builtin grammars, and the number of results to from the recognizer (n-best results). | | |
| Number/ **Digits** | Captures a string of digits using the built-in digits grammar.<br>**DTMF**: string of touchtones.<br>**Voice**: the number is **spoken one digit at a time** | | DDDD |
| Number/ **Number** | Collects a number using the built-in number grammar<br>**DTMF**: use optional * for decimal point<br>**Voice**: Any positive, negative, or decimal value | | DDD[.MMM] |
| Number/ **Phone** | Collects a phone number using the built-in phone grammar | | |
| Menu/**#_ Option_ Menu** | The **[X]_Option_Menu** voice elements provide menus that support from 2 to 10 options. Menu voice elements are similar to the Form voice element except the number of options are fixed, all grammars are defined in the voice element itself, and a separate exit state for each option. | value<br>value_confidence | |
| Menu/ **YesNo_ Menu** | The **YesNo_Menu** voice element presents a yes/no menu and uses the boolean builtin grammar.<br>**DTMF**: 1 is used for yes and 2 for no.<br>**Voice**: "yes", "no" and other synonyms (a complete list of synonymous utterances is voice browser dependent). | value<br>value_confidence | yes/no |
| <CollectType> _With_ Confirm | Collects one of the above types during the Collect stage.<br>And then executes a YesNo_Menu during the Confirmation stage. | value<br>value_confidence<br>confirm_confidence | |

## Chapter 4 Review Questions

1. How do you ensure that a caller's private data is not stored in the Activity log?

   ENABLE SECURE LOGGIN TRUE

2. Which of these elements require the caller to terminate their entry with #. Explain.
   - Digits?
   - Number?
   - Currency?

3. What is the interdigittimeout?

   TIME BETENEN TOUCHTONES (DTMF)

4. How do you configure the interdigittimeout for the entire application? For one element?

   PROPERTIES /ROOT DOC SETTING /"INTERDIGITSTIMEOUT

YES, NO MENU

5. In a Digits element, how do you configure the gateway to wait for the termination character if the caller enters the maximum number of DTMF tones?

MAXDIGITS+1 OR NOT USE

   TERMINATES TIMEOUT TERMTIMEOUT

6. In a Menu element, what must be done to allow the # to be collected as a valid menu option?

   # TERMCHAR WITH SPACE [ ]

7. If an error occurs in the application when you are debugging it, where should you look to determine which element had the error? Where should you look for details of the error that occurred?

   C:\CISCO\CALLSTUDIO\ECLIPSE\PLUGINS\ COM.AVDUM(ERR.STUDIO.DEBUGRUNT AVDUM_HOME\APPLICATION\MYAPP\LOGS

8. What is the difference between the Digits element, Number element, Currency element in terms of what callers can enter as DTMF?

   DIGITS - # KEY CAUSE A NOMATCH EVENT
   NUMBER - ANY NUMBER OF DIGITS AFTER # KEY (UNTIL PRESS #)
   CURRENCY - WAITS FOR MAX 2 DIGITS AFTER KEY
   NUMBER ELEMENT

a pasta application só aparecerá após o debug da aplicação

depois cap. 5 - 7

## Chapter 4 Programming Lab

Rename the Insurance project to **MyApp**

### Part 1: Collect an Insurance Account Number from the caller

1. On page1, use a **DigitsWithConfirm** element between the **pWelcome** and **mInsType** to collect and confirm the caller's account number.
   A. Name the DigitsWithConfirm element **getAcct**
   B. Audio Tab:
      a. **Digits Initial**: *Tell me your 4 to 7 digit insurance account number. Or key it in followed by the pound sign.*
      b. **Confirm Initial**
         - **Audio item1**: *Did you enter*
         - **Audio item2**: select Say it Smart
           **Data**: {Data.Element.getAcct.value}
           **Type**: Digit-by-Digit
         - **Audio item3**: If that's correct say yes or press 1. Otherwise say no or press 2.
      c. **Confirm > Disconfirm1:** *Let's try again.*
      d. **Confirm > Disconfirm2:** *Please try one more time.*
      e. **Confirm > Disconfirm3:** *I just can't get it right.*
      f. **End > Yes** *Great.*

   C. Settings Tab:
      - Digits Confidence Level **0**
      - Confirmation Confidence Level **0**
      - Min Digits: **4**
      - Max Digits**: 8**
      - VoiceXML properties: Name: **interdigittimeout** Value: **2s**

2. Connect the **getAcct →done** exit state to the **mInsType** element.

3. Connect the **getAcct max_nomatch, max_noinput, and max_disconfirmed** exit states to a new Page Connector that goes to the **CSR** page.

4. Save and test your application.

5. Save, deploy, test.

**Continue to Part 2.**

**Part 2: On the GetPayment page, add code to collect a payment amount from the caller:**

1. Connect **ynMakePayment → yes** exit state to a CurrencyWithConfirm element (named **getPaymentAmount)**
   A) Name the Currency element **getPaymentAmount**
   B) Settings – set both the Currency Confidence and Confirm Confidence Levels to 0
   C) Audio Tab
   i. **Currency Initial**: *Please tell me the amount you would like to pay toward your {Data.Element.mInsType.value} balance. Or key it in followed by the pound sign.*
   ii. **Confirm Initial**:
   - Audio item1*: Would you like to pay*
   - Audio item2: (Say It Smart)
     Data: {Data.Element.getPaymentAmount.value}
     Type: Currency
   - Audio item3*: If this is correct, say yes or press 1. If not correct, say no or press 2.*

   iii. **Confirm > Disconfirm1:** *I'm sorry. Please try again.*
   iv. **Confirm > Disconfirm3:** *I'm sorry, I just can't get it right.*

2. Connect the **getPaymentAmount → max_nomatch, max_noinput, max_disconfirmed** exit states to the Page Connector that connects to the **CSR** page.

3. Connect the **getPaymentAmount → done** exit state to the **pHoldPayments** element.

4. Modify the CVP Subdialog Return that **pHoldPayments** connects to by adding 2 more return values containing the account number and the payment amount.
   - Caller Input: payments
   - External VXML0: insType={Data.Element.mInsType.value}
   - External VXML1: account={Data.Element.getAcct.value}
   - External VXML2: amount={Data.Element.getPaymentAmount.value}

**Challenge (Only if you are done early)**

1. Before asking the caller for the payment amount, use a Number or NumberWithConfirm element (named **getNumPmts**) to collect the number of monthly payments the caller would like to schedule. Speak it back to the caller.

2. Use a Date element (not Date with Confirm) named **getPayDate** to collect the date to start payments. The caller must say the full date (example, January 31st 2009) or must key it in as YYYYMMDD (example, 20101225) format. And then use a separate Audio element to speak the date to the caller using the Date Say It Smart (YYYYMMDD). NOTE - when testing you must say or enter a FULL date, otherwise a Say It Smart exception occurs. Consult the Activity and Error Logs.

[End of Chapter 4]

B. Connect the **valid** exit state to a database element (**dbWithdraw**) to perform the withdrawal:
**Type: UPDATE**
**JNDI: rx**
**Query: update account**
**set balance1 = balance1 – {Data.Element.getWithdrawAmt.value}**
**where acctnum = {Data.Element.getAcct.value}**

NOTE -This does NOT create any ariables ata it just updates the DB.

3. Now select the updated information from the DB and speak it by performing a Copy/Paste of the **dbAcctInfo** element. Rename it **dbNewBalance**.

4. Then speak the value of {**Data.Element.dbNewBalance.balance1}** as Currency to the caller. Tell them that a check for {**Data.Element.getWithdrawAmt.value}** should arrive shortly.

5. Connect to **pGoodbye**.

## Challenge (Only to be attempted if you are done early)

If a caller has 2 accounts, ask the caller from which account to make the withdrawal and update the associated data in the database. Speak the new balances.

Note that 2222 and 3333 only have one account. The account2 and balance2 values are NULL in the database and the Select statement will return nothing. VxmlServer will not create or overwrite existing values for those Element Data variables.

**1.** Change the **dbAcctInfo** element:
- Change the SQL query to:

**select * from account where acctnum={Data.Element.getAcct.value}**

- In the Data tab of this element, create the variables **account2** and **balance2** (Before) and set their values to **;**

2. After the Database element, use a Decision to test if the caller has a second account:

If **{Data.Element.dbAcctInfo.account2}** Does Not Equal **;**
then they have a second account

3. If the caller has a second account, speak each account name and its balance:
- The balance of your {Data.Element.dbAcctInfo.account1} account is {Data.Element.dbAcctInfo.balance1} (as Currency).

- The balance of your secondary {Data.Element.dbAcctInfo.account2} account is {Data.Element.dbAcctInfo.balance2} (as Currency).

- Element data for Element **dbAcctInfo** named **account1 Exists** ;
- **AND** Element data for **dbAcctInfo** named **account1 does not Equal (String) the Constant String** ;
- Assign the Exit State: **valid**
- Set the Otherwise Exit State: **invalid**

8. For the **invalid** path:
   - Connect to an Audio element (named **pInvalid**) that tells the caller the account number is **not valid**.
   - Connect to a separate element (named **pGoodbye**) that says goodbye to the caller.
   - Return the call back to ICM and set Caller Input to **done** so ICM releases the call. Name this element **CVPSubdRetDone.**

9. For the **valid** path: Connect to an Audio element (named **pBalance).**
   - Speak the name of the account (Savings or Checking) from the variable {Data.Element.dbAcctInfo.account1}
   - Speak the balance as Currency from the variable {Data.Element.dbAcctInfo.balance1}
   - Connect to **pGoodbye**.
   - Connect to the **CVPSubdRetDone** element

10. Create a new page and rename it **CSR**.
    - Delete your page2 Entry connector on the CSR page.
    - Open the **Insurance** application you've worked on and copy the contents of the **CSR** page and paste it onto your **CSR** page.

11. On page1, Connect all MaxNomatch and MaxNoinput exit states to a page connector that points to the CSR page.

12. Save and test the application.

**Part II**

1. Collect the currency amount the caller wants to withdraw (name this element **getWithdrawAmt**) and repeat the amount back to the caller. Use the prompt *"enter the amount they'd like to withdraw followed by the pound key"*

2. (**testAmount**) After the caller confirms the amount, use a Decision element to test:
   if the amount entered by the caller {Data.Element.getWithdrawAmt.value}
   <=
   The balance in the DB {Data.Element.dbAcctInfo.balance1}
   Exit State: **valid**
   **Otherwise Exit State: invalid**

   A. Connect the **invalid** exit state to an Audio (**pAmtTooHigh**) element that tells the caller they can not withdraw more than the balance dbAcctInfo.balance1 (as Currency). Connect back to the **getWithdrawAmt** element.

## Chapter 7 Programming Lab

Create a new MyApp application to allow CVP Bank callers to withdraw funds by phone. Use the 'rx' Database to provide the caller with their balance and to perform the withdrawal.

In the MySql Command client, enter the following command to view the table:

mysql> **select * from account;**

| **acctnum** | **account1** | account2 | **balance1** | balance2 |
|---|---|---|---|---|
| **1111** | **SAVINGS** | CHECKING | **1000** | 2000 |
| **2222** | **SAVINGS** | NULL | **2200** | NULL |
| **3333** | **CHECKING** | NULL | **3300** | NULL |
| **4444** | **SAVINGS** | MONEY MARKET | **4400** | 6000 |
| **5555** | **SAVINGS** | CHECKING | **1000** | 2000 |

### Part I

1. Create a new CVP Project, named **MyApp**. Configure the Project / Properties / General Settings - Gateway Adapter.

2. Use a CVP Subdialog Start element.

3. Create a 2nd page named CSR and copy the **CSR** page code from another application. All **max_nomatch** and **max_noinput** paths will connect to the CSR page.

4. Welcome the caller to **CVP Bank Withdrawals**.

5. Collect the caller's 4-digit bank account number (name the element **getAcct)** and repeat it back to the caller.

6. Use a Database element (named **dbAcctInfo**) to get the account name and its balance.

   A. Settings tab:
   **Type: Single**
   **JNDI: rx**
   **Query: select account1, balance1 from account where acctnum = {Data.Element.getAcct.value}**

   B. In the Data tab, create and initialize the **Element Data**:
   - Name: **account1** Value: ; Create: **Before** (Press Add)
   - Name: **balance1** Value: ; Create: **Before** (Press Add)

7. Use a Decision element (named **testAcct** ) to check if:

## Chapter 7 Review Questions

1. In a database element, if you select 'Single' where is the returned data stored?

2. In a database element, if you select 'Multiple' where is the returned data stored?

3. What is a Best Practice when using a database element to perform a 'Single' select so variables are initialized?

4. What is the location and name of the Tomcat configuration file that is modified to allow using the database element? And where must the JDBC driver be stored?

5. Same question for Studio Debugger's Tomcat?

6. How do you configure the gateway to allow 60 seconds for your database or web services element to execute? What occurs if the gateway times out?

7. How do you have the gateway play music to the caller while it's waiting for your database element to complete?

## **Email Element** (Folder: Elements/Notification/)

1. The Studio Email element allows the VXML Server to send email using the **JavaMail** package supplied by the application server.

2. The message can include attachments from the VXML Server system. Attachments that do not exist will be skipped but the message will still be sent.

3. The application server must be configured to set a JNDI datasource for mail sessions (see below).

4. The **to** (or **toList**) field must be defined. Email addresses are not verified for syntax or validity. The **toList**, **ccList** and **bccList** settings must refer to session data variables that hold a list of email addresses retrieved from a Database element with setting **Type:Multiple**).

5. Tomcat configuration to enable use of the Email element:

   A. Download the JavaMail package from:
   **java.sun.com/products/javamail/downloads/index.html**

   B. Download JavaBeans Activation Framework. from
   **http://java.sun.com/products/javabeans/glasgow/jaf.html**

   C. Copy **JavaMail .jar** and **activation.jar** to **VXMLServer\Tomcat\common\lib**

   D. Configure Tomcat JNDI by adding this to the VXMLServer\Tomcat\conf\context.xml file

```
<Context>
  <!-- Default set of monitored resources -->
  <WatchedResource>WEB-INF/web.xml</WatchedResource>
  <Manager pathname="" />
   <Resource name="mail/myEmail" type="javax.mail.Session"
             mail.smtp.host="myEmailServer.mail.com" />
<\Context>
```

   NOTE - In Studio, enter the JNDI name ***MyEmail*** - do not include the "mail/"

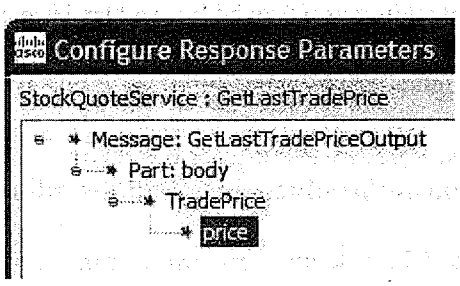
   E. Restart the **Cisco CVP VXML Server** Service

d) **Some** parameters cannot hold a value (they show "N/A" as their type). For example "disable_logging", if it exists it disables logging.
e) Only variables with a type can hold a value. The value will be validated as you type and when you validate the project.

f) **Repeatable** settings have an index in brackets (such as the "item" parameter above). To add/delete parameters, **right-click** and choosing **"Add PARAM_NAME"** or **"Delete PARAM_NAME"**

**10. Configuring Response Parameters**

a) Response parameters (data sent back by a web service) are handled in much the same way as request parameters.




b) You must specify whether each parameter should be stored in Element or Session data.

c) The text input field is used to specify the variable name to create.

d) No type-checking is performed in this dialog; the response parameter type is listed only for convenience.

e) The most significant difference between this dialog and the "Configure Request Parameters" dialog is that parameters marked as required do not need to be configured. Any parameter not configured in this dialog **will simply not be stored in element or session data at runtime**; if it is present in the web service's response, it will be ignored.

RENAME THE WSDL (LOADED AGAIN) (CHANGE ID)

The name of the file containing the wsdl used in an element can be found in this file **<appname>/callflow/pages/<pagename>/elements/<wsdl_element_name>.**

If the wsdl information changes at a later time, the Studio version of the wsdl file must be modified and the Studio application must be redeployed. You should take precautions to **back up the Studio application** before attempting this.

7. **Configure Web Service Call**
   a) **Service** The service to invoke (its namespace lists in parenthesis). Generally, WSDL files only define a single service so this list may have just one item.

   b) **Port** Specify the port to use to connect to the web service at runtime.

   c) **Operation** Specify the operation to execute against the previously-selected service.

   d) **Request:** Click the "Configure" button to configure data to send to the web service.

   e) **Response** Click the "Configure" button to specify where to store return values.

   f) **Store Full Response XML** Check this box to store the full XML response in element data at runtime, for later processing by your own custom code,

8. **Runtime Settings**

   a) **Connect Timeout:** The number seconds to wait for a response to a request at runtime, before following the **"error"** exit state.

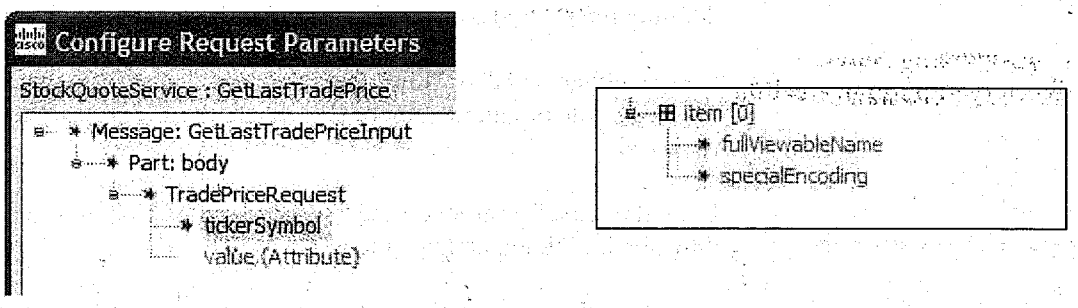
   b) **Requires HTTP Authentication** Select this to use HTTP authentication when accessing the web service at runtime. **Username, Password** are then available

   c) **Use Proxy** Check this box if you would like a proxy to be used when accessing the web service at runtime. **Proxy Host**, **Proxy Port** are then available

9. **Configuring Request Parameters**

   a) The following dialog is displayed and its contents are pre-populated with parameters that the loaded WSDL specifies. Displays in a tree format, displays required * and repeatable + settings.

   b) **Optional** settings are **greyed-out** (like "value" below), and can be added by right-clicking on it and choosing "Add PARAM_NAME":




   c) **Each** parameter has a type, such as string, integer, or float.

## Web Services Element

1. The **Web Service element** provides interaction with remote web services, using Web Service Definition Language (WSDL) for service definitions and SOAP for message encapsulation.

2. This Web Service element is designed to work with the following technologies:
   a. **WSDL 1.1** (using namespace http://schemas.xmlsoap.org/wsdl/)
   b. **SOAP 1.1** encoding (using namespace http://schemas.xmlsoap.org/soap/encoding/)
   c. XML schemas (using namespace http://www.w3.org/2001/XMLSchema)

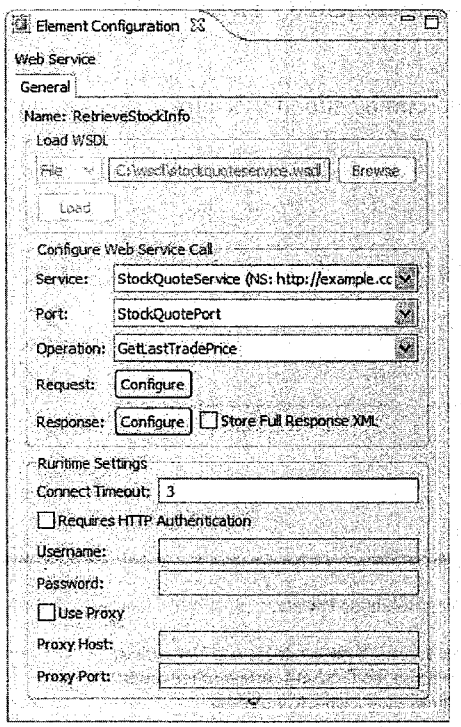
3. This element includes built-in support for 1-dimensional arrays (that is, sequences). To parse n-dimensional arrays (where n is greater than 1) in web service response messages, use the **"Store Full Response XML"** checkbox and write custom Java code.

4. **Exit States**
   a. **done:** Successful response from the web service.

   b. **error** A runtime error occurred. Examples, web service cannot be reached, the "Connection Timeout" expires before a response arrives, or receiving unexpected data from the service.

   c. **fault** This exit state is only present when the loaded WSDL specifies a possible **fault** message and the web service is **successfully contacted** at runtime, but it responds with its fault message.

5. **Element Data**
   a. **response_xml** – This variable is only created when you select the "Store Full Response XML" checkbox (and it is not then logged by default). This could be for debugging/logging or for parsing n-dimensional arrays returned from the web service using some custom Java code.




6. **Configuration:**

**Load WSDL > WSDL Location**

In order for the Web Service element to be configurable, a WSDL file defining the desired web service must first be loaded.

Choose either "URI" or "File" from the drop-down, then either browse for a local file or enter a remote URI where the WSDL can be retrieved.

Click the "Load" button to initiate Cisco Unified Call Studio's download, caching, and parsing of the WSDL.

Once WSDL is loaded, the other configuration options become available.

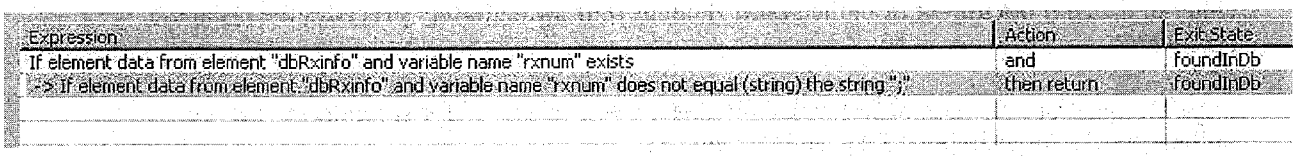
## Programming Tip – Timers and FetchAudio Music

Since a database lookup may take a while, be sure to configure fetching properties beforehand, either in the **root document** or in the **audio element that immediately precedes** the Database element.

Time Designations are non-negative real numbers, followed by "s" for seconds, or "ms" for milliseconds". For example, "3.5s" and "800ms".

| | |
|---|---|
| **fetchaudio** | The URI of the audio to play while waiting for a document to be fetched. There are no fetchaudio properties for audio, grammars, objects, and scripts. The fetching of the audio clip is governed by the audiofetchhint, audiomaxage, audiomaxstale, and fetchtimeout properties in effect at the time of the fetch. The playing of the audio clip is governed by the fetchaudiodelay, and fetchaudiominimum properties in effect at the time of the fetch. |
| **fetchaudiodelay** | The time interval to wait at the start of a fetch delay before playing the fetchaudio source. Default: platform-dependent, e.g. "2s". The idea is that when a fetch delay is short, it may be better to have a few seconds of silence instead of a bit of fetchaudio that is immediately cut off. |
| **fetchaudiominimum** | The minimum time interval to play a fetchaudio source, once started, even if the fetch result arrives in the meantime. Default: platform-dependent, e.g., "5s". The idea is that once the user does begin to hear fetchaudio, it should not be stopped too quickly. |
| **fetchtimeout** | The timeout for fetches. Default: platform-dependent. The VoiceXML event **error.badfetch** is thrown when this timer expires. |

19. After the Database element executes, use a Decision element to test if the element data exists AND if its value DOESN'T EQUAL the dummy assigned in the data tab.



| Expression | Action | Exit State |
|---|---|---|
| If element data from element "dbRxinfo" and variable name "rxnum" exists | and | foundInDb |
| -> If element data from element "dbRxinfo" and variable name "rxnum" does not equal (string) the string "; | then return | foundInDb |

**Other Database Types**

1. When selecting the Database element type **Multiple** the information is stored into a **Java object** of type **ResultSetList** which requires custom Java to access the rows and columns of the data.

   - If you prefer not to use custom java, then most SQL databases have a way to limit the number of rows returned.

   - This example uses an incrementing counter to select successive records from the database. Since it is only selecting a single record, the result is stored into Element data.
     **select * from rxtable limit {Data.Element.ctDataCount.count}, 1**

2. The **Update** statement does not create any variables, although an entry will be logged to the Activity log denoting the number of records updated.

| **Update** |
|---|
| Rx |
| update rxtable set rxquantity = rxquantity-1 where rxnum={Data.Element.get_rxnum.value} |

3. The **Insert** statement does not create any variables, it just adds a new row of data into the database (no data is returned):

| **Insert** |
|---|
| Rx |
| insert into rxpickuptable values (<br>{Data.Element.get_rxnum.value},<br>'{Data.Element.db_rxinfo.rxprice}',<br>'{Data.Element.get_pickuptime.value}',<br>'{Data.Element.get_location.value}' ) |

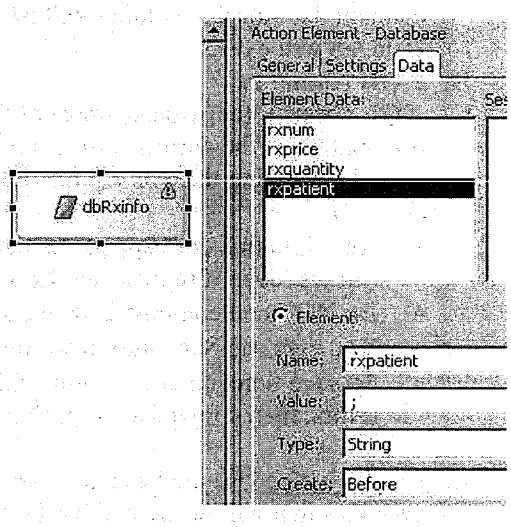
4. Stored Procedures can be invoked using the Single or Multiple type (based upon how many rows of data the procedure returns) and using the EXEC statement.

16. Use a **Decision Element** to test if ElementData for Element: **dbRxValidate** Data: **numrecs** is greater than 0.

17. If so, create another Database element named **dbRxInfo** to retrieve the RX information.

| | |
|---|---|
| *Type | Single |
| *JNDI Name | rx |
| *SQL Query | select * from rxtable where rxnum = {Data.Element.getRxnum.value} |

Use the DB Element's **Data tab** to initialize the element data this element will create at runtime.




18. **Database Notes**

A. If the database doesn't respond the VXML Server throws a Java exception.

B. If a SELECT query does not match any rows, no variables are created, you can use the Decision Editor to check if the variable EXISTS or if initialized variables contain the default/dummy value assigned (e.g., semi-colon).

C. If a SELECT query returns data, but a column has a **null** value in the database, VXML Server does not create (or does not overwrite a previous value of) that variable.

D. **NOTE**: The Oracle database returns column names UPPER CASE – so all variables are created with UPPER CASE names.

E. **Best Practice** Use the Database's **Data tab** to create and initialize the Element Data variables that will be created by the database query. Assign a default value that will never be returned, and create them **BEFORE**.
- After executing a Select statement, use a Decision element to test if the variable contains the default value. Use this to determine if a value was or was not returned from the database.
- This also makes the element data names appear in the Substitution window.

14. In the Call Studio application name the DB element **dbRxValidate**

| | |
|---|---|
| *Type | Single |
| *JNDI Name | **rx** |
| *SQL Query | select count(*) as numrecs from rxtable where rxnum = {Data.Element.getRxnum.value} |

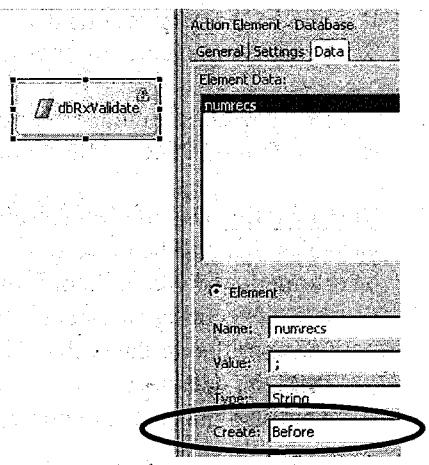
A. **Type: Single** - Retrieve the first row that matches the query. Element data with the column names is created for each column returned. If a column has a NULL value in the database, the element data for that column will NOT be created and the dummy value you initialized in the Data tab will exist.
B. **JNDI Name: rx**
C. **SQL Query:** (copy/paste most of this from MySql Command Line Client): You could optionally use the **count(*)** function in the select statement to determine the number of records matching the query. This creates Element data named ***'numrecs' >= 0.***

- Any field that corresponds to a character field in the database must be set between **single quotes** (even if represented by a Substitution tag).
- Do NOT end commands with a semi-colon.
- You may use Substitution by invoking the Substitution Tag Builder
- To save time, enter the command to the SQL database to ensure there are no syntax errors, then copy/paste it into Studio.

15. **Best Practice** Use the Database element's **Data tab** to create and initialize the Element Data variables that will be created by the query. Assign a default value that will never be returned, and create them **BEFORE**.
    - This makes the element data names appear in the Substitution window.
    - This allows you to use a Decision element to test if the variable contains the default value – this aids in determining if a value was or was not returned from the database.

**NOTE**: Oracle Database returns column names UPPER CASE – so all variables must be created with UPPER CASE names.

Query:
SELECT count(*) as numrecs
FROM RXTABLE
WHERE
RXNUM={Data.Element.getRxnum.value}



**Classroom Example**

1. The classroom is configured with the open source MySql RDBMS, with the following database and tables.

2. Follow the menu path **Start / Programs / MySql / MySql Command Line Client to access the database.**

3. password> root
4. mysql> use rx; 'rx' is the name of the database
5. mysql> **show tables;** display a list of all tables in this database

   RXTABLE
   RXPICKUPTABLE
   ACCOUNT

6. **mysql> desc rxtable;** show the column names and types

| | |
|---|---|
| RXNUM | INT (4) |
| RXPRICE | VARCHAR (12) |
| RXQUANTITY | INT (4) |
| RXPATIENT | VARCHAR (40) |

7. **mysql > select * from rxtable;** display all columns and rows of the table

| rxnum | rxprice | rxquantity | rxpatient |
|---|---|---|---|
| 1111 | 11.11 | 11 | JONES |
| 2222 | 22.22 | 22 | SMITH |
| 3333 | 33.33 | 33 | WAYNE |
| 4444 | 44.44 | 44 | CLIFTON |

8. **mysql > select * from rxtable where rxnum=1111;**

| rxnum | rxprice | rxquantity | rxpatient |
|---|---|---|---|
| 1111 | 11.11 | 11 | JONES |

9. **mysql > select * from rxtable where rxnum=1234;**

   Empty Set

10. **mysql > select count(*) as numrecs from rxtable where rxnum=1111;**

| numrecs |
|---|
| 1 |

11. **mysql > select count(*) as numrecs from rxtable where rxnum=1234;**

| numrecs |
|---|
| 0 |

12. **mysql > update rxtable set rxquantity=rxquantity-1 where rxnum=1111;**

    Query OK; 1 row effected

13. **mysql > select * from rxtable where rxnum=1111;**

| rxnum | rxprice | rxquantity | rxpatient |
|---|---|---|---|
| 1111 | 11.11 | 10 | JONES |

**The Studio Database Element Settings**

Configure each Studio Database element to perform one interaction with an RDBMS system.

1. **Type**
   - **Single**. Execute a SQL SELECT statement that returns only a single row, the first row that matches the query.
     - **Element data** variables will be created with the **names of the columns** returned.
     - No element data will be created for those columns that with a NULL value

     SELECT **BALANCE, LASTDEPDATE** FROM ACCOUNTS WHERE ACCT=1111

   - **Multiple**. Execute a SQL SELECT that returns multiple rows into a Session data variable (object) of Java class ResultSetList.
     - Specify a name for the resulting Session data object in the Setting **Session Data Key**
     - If no rows are returned, the ResultSetList object in Session data will be empty.
     - Custom Java must be written to access its values of the ResultSetList object.
     - For detail about the ResultSetList data structure, refer to the Cisco CVP Javadocs.

     SELECT **ACCNAME, BAL** FROM ACCOUNT

   - **Insert**. Execute a SQL INSERT command that inserts information into the database.
     - No data is returned. No data is created.

     INSERT INTO BILLPAY (ACCTNUM, AMT) VALUES **( '{Data.Element.get_acct.value}', {Data.Element.get_amt.value} )**

   - **Updates**. Execute a SQL UPDATE command that updates information in the **database**.
     - No data is returned. No data is created.

     UPDATE ACCOUNT SET BALANCE = BALANCE - {Data.Element.getAmt.value} WHERE ACCTNUM = '{Data.Element.getAcct.value}'

2. **JNDI Name** Enter the JNDI name (alias) as configured in Tomcat's context.xml file.

3. **Query** Enter the SQL query, including Substitution tags if necessary. Do NOT end with semi-colon in Studio. Queries may contain sub-queries.

3. Restart the Windows service named **Cisco CVP VXML Server** service.

4. **Studio Debugger Configuration**

   To use the Database element with the Studio Debugger do the following:

   - Modify the context.xml file located in C:\Cisco\CallStudio\eclipse\plugins\com.audiumcorp.studio.debug.runtime_x.y.z\CATALINA_HOME\conf

   - Copy the Java JDBC driver into C:\Cisco\CallStudio\eclipse\plugins\com.audiumcorp.studio.debug.runtime_x.y.z\CATALINA_HOME\common\lib

## Configuring Tomcat JNDI for the MySql Database

Tomcat Configuration for Oracle, SQL Server, and MySQL.

1. Modify this file: C:/Cisco/CVP/VXMLServer/Tomcat/conf/context.xml

```
<Context>
  <!-- Default set of monitored resources -->
  <WatchedResource>WEB-INF/web.xml</WatchedResource>
  <Manager pathname="" />
      Enter one of the following here
</Context>
```

   A. Oracle: set the <Resource> tag as below:

```
<Resource auth="Container"    name="jdbc/rx"
 url="jdbc:oracle:thin:@127.0.0.1:1521:mysid"
driverClassName="oracle.jdbc.OracleDriver"
type="javax.sql.DataSource"  username="myuser"  password="pwd"
maxActive="4"  maxIdle="2"   maxWait="5000" removeAbandoned="true"
 removeAbandonedTimeout="5" logAbandoned="true"
 validationQuery="select 1 from dual"/>
```

   B. SQL Server: set the <Resource> tag as below:

```
<Resource  name="jdbc/rx " type="javax.sql.DataSource"  auth="Container"
factory="org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory"
driverClassName=" com.microsoft.sqlserver. jdbc.SQLServerDriver"
url=" jdbc:sqlserver://localhost:1433;databaseName=rx;SelectMethod=cursor"
username="training"    password="training"
maxIdle="2"   maxWait="5000"  maxActive="4"/>
```

   C. MySQL configuration:

```
<Resource auth="Container"
 name="jdbc/rx"
 url="jdbc:mysql://localhost:3306/rx"
 driverClassName="com.mysql.jdbc.Driver"
 type="javax.sql.DataSource"
 username="root"   password="root"
 maxActive="4"
 maxIdle="2"
 maxWait="5000"
  removeAbandoned="true"
  removeAbandonedTimeout="600"
  validationQuery="select 1 from dual"/>
```

2. Install the RDBMS-specific JDBC driver (Example, **mysql-connector-java-3.1.13-bin.jar)** into the Tomcat directory:

   **C:/Cisco/CVP/ VXMLServer/Tomcat/common/lib/**

- **maxWait** - The maximum number of **milliseconds** that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. Default: -1 (infinite)

3. Some additional properties handle connection validation:

   - **validationQuery** - SQL query that can be used by the pool to validate connections before they are returned to the application. If specified, this query MUST be an SQL SELECT statement that returns at least one row.
   - **validationQueryTimeout** - Timeout in seconds for the validation query to return. Default: -1 (infinite)
   - **testOnBorrow** - true or false: whether a connection should be validated using the validation query each time it is borrowed from the pool. **Default: true**

4. Another optional feature is the removal of abandoned connections. A connection is called abandoned if the application does not return it to the pool for a long time. The pool can close such connections automatically and remove them from the pool. This is a workaround for applications leaking connections.

The abandoning feature is disabled by default and can be configured using the following properties:

- **removeAbandoned** - true or false: whether to remove abandoned connections from the pool. Default: false
- **removeAbandonedTimeout** - The number of seconds after which a borrowed connection is assumed to be abandoned. Default: 300
- **logAbandoned** - true or false: whether to log stack traces for application code which abandoned a statement or connection. **This adds serious overhead**. Default: false

## The Database Element (Folder: Elements/Integration/Database)

1. The **Database** element provides an interface to SQL databases where the VXML Application Server can be configured to interface to it using **Java Naming and Directory Interface (JNDI).**

2. **JNDI** is part of the Java platform that provides a unified interface to multiple naming and directory services such as databases and email servers.

3. Tomcat performs **database connection pooling** - recycling and reusing already existing connections to a DB- which is more efficient than opening new connections.

### Configuring Tomcat JNDI Database Interface

1. VXMLServer's Tomcat is configured by adding the bold text to the file **C:/Cisco/CVP/VXMLServer/Tomcat/conf/context.xml**

```
<Context>
  <!-- Default set of monitored resources -->
  <WatchedResource>WEB-INF/web.xml</WatchedResource>
  <Manager pathname="" />
    <Resource auth="Container"
     name="jdbc/rx"
     url="jdbc:mysql://localhost:3306/rx"
     driverClassName="com.mysql.jdbc.Driver"
     type="javax.sql.DataSource"
     username="root"   password="root"
     maxActive="4"
     maxIdle="2"
     maxWait="5000"                          (milliseconds)
     removeAbandoned="true"
     removeAbandonedTimeout="600"        (seconds)
     validationQuery="select 1 from dual"/>
  </Context>
```

2. Tomcat data source configuration properties are as follows:

- **driverClassName** - Fully qualified Java class name of the JDBC driver to be used.
- **username** - Database username to be passed to our JDBC driver.
- **password** - Database password to be passed to our JDBC driver.
- **url** - Connection URL to be passed to the JDBC driver. (For backwards compatibility, the property `driverName` is also recognized.)
- **initialSize** - The initial number of connections that will be created in the pool during pool initialization. Default: 0
- **maxIdle** - The maximum number of connections that can sit idle in this pool at the same time. Default: 8
- **maxActive** - The maximum number of connections that can be allocated from this pool at the same time. Default: 8

# Chapter 7

# The Database and Web Services Elements

- The Database Element
- Configuring Tomcat for JNDI
- VoiceXML Properties
- The Web Services Element

5. Connect **testAmount → valid** to a new Math element you create (named **mthPayment**) to subtract the payment amount from the Session variable created earlier **insBalance**, store the result back into the same Session Data variable **insBalance**

   Settings
   Type: **Session**
   Name: insBalance
   Expression: {Data.Session.insBalance} – {Data.Element.getPaymentAmount.value}

6. Connect **mthPayment** to a new Audio element (named **pNewBalance**) that tells the caller:
   - *the payment has been credited and the new insurance balance is*
   - Say it Smart: {Data.Session.insBalance} as Currency.

7. Connect to the existing **pGoodBye** element.

8. Delete the orphaned elements **pHoldPayment** and **CVPSubdialogReturnPayments**.

[End of Chapter 6]

## Chapter 6 Programming Lab

1. Modify the Insurance application as follows. Rename **Insurance** to **MyApp**.

2. On page 1, use an Application Modifier element (named **declareVariables**) immediately after the CVP_Subdialog_Start element to create a Session Data variable:
   - Name: **insBalance**
   - Value: **1000.00**
   - Create: **Before**
   - Click the button **ADD**

3. On the **getPayment** page, after collecting the payment amount in the **getPaymentAmount** element, use a Decision element (named **testAmount**) to ensure the getPaymentAmount.value is less than the Session variable insBalance:

| Element Data | Operator: <= | Session Data | Exit State: **valid** |
|---|---|---|---|
| getPaymentAmount<br>value | | insBalance | Otherwise Exit State:<br>**invalid** |

4. Connect **testAmount → invalid** to a new Counter element (named **ctNumTries**). The **ctNumTries** element will be used to give the caller 3 tries to enter a valid amount. Use the default Settings, so it increments a counter.

   A) Connect **ctNumTries→ done** exit state to a new Decision element (named **testNumTries**). The **testNumTries** element should check if the counter is less than 3 to allow the caller to try again.

| Element Data | Operator: < | Constant Number | Exit State: **again** |
|---|---|---|---|
| ctNumTries<br>count | | 3 | Otherwise Exit State:<br>**max** |

   B) Connect **testNumTries → again** exit state to a new Audio element (named **pTryAgain**) that says:
      - *The amount entered was larger than the amount due.*
      - *{Data.Session.insBalance}* (spoken using Say it Smart, currency)
      - *Please try again.*

   C) Connect **pTryAgain** to the **getPaymentAmount** element.

   D) Connect **testNumTries → max** exit state to a new Audio element (named **pTooManyErrs**) that says: *the amount entered is still not valid.* Connect **pTooManyErrs** to a Page Connector to the **CSR** page.

## Chapter 6 Review Questions

1. What are a few differences between Element data and Session data variables?

   ELEMENT DATA = NOT CHANGE THE VALUE (VARIABLE LOCAL)
   SESSION = YOU CAN CHANGE THE VALUE ( GLOBAL

2. Where can you create variables? What is a Best Practice for where you should declare and assign variables?

   BEST PRATICE - USE APP MODIFIER
   ANY ELEMENT DATA TAB. AT THE BEGINNING TO DECLARE ALL VARS

3. Which element do you use to subtract a paymentAmount from an insuranceBalance and store the result back into the insuranceBalance variable?

   MATH

4. What are some limiting factors about the Counter element?

   CAN'T INCREMENT SAME COUNTER SOMEWHERE ELSE IN APP, CAN'T RE-SET THE COUNTER

5. What would be a workaround for these limitations?

   CREATE A SESSION VARIABLE 'S COUNTER' AND USE MATH ELEMENT TO INCREMENT IT.

6. In a Data tab, after creating a variable, you decide to modify its initial value. What must you remember to do so that this modification is actually stored into the Studio application?

   PRESS UPDATE BUTTON.

- Since there are often multiple entry points into an Element Group, the exit states that match these entry points are differentiated by using the following naming convention: **ElementName:ExitStateName**

- Optionally, the entry points of an Element Group can be consolidated into a single exit state from the Group Start element. This can be used to easily map all entry points to one element within the Element Group. **Right-click** the Group Start and select **Entry Points > Single**

- Unconnected **exit states** of elements within the Element Group are available as exit states for the **Element Group shape** in the call flow editor.

- The exit states are accessible by right-clicking on the Element Group shape as would be done with any call flow component. The Element Group exit state naming convention is **ElementName:ExitStateName**

**Saving an Element Group**

A. **Element Groups can be saved as Element Group Templates (EGTs).** Once an Element Group has been saved as an EGT, it displays in the Elements pane.

B. When an EGT is saved, a single file with an "**.egt**" extension is created. Since EGTs are templates, changes (including the addition of elements, configuration modifications, etc.) made within instances of an EGT do not change the reusable template file.

C. Saving an Element Group into an EGT:
   a) **Right-click** on an Element Group and select **Save As**.
   b) A **wizard** will pop up that includes several options related to saving the EGT.
   c) Once the EGT is saved, it can be viewed under Element Groups in the Call Studio Elements View. You can use this group in any application.

## Element Groups (Reference)

1. An **Element Group** allows you to group elements, and then treat the new group as a reusable call flow component.

2. Element Groups can be saved as templates in the Elements pane and can be dragged into other applications for reuse.

3. **Creating an Element Group**
   A. Element Groups are created by selecting elements that should be part of the new group.

   B. Now **right-click** and choose **Group Elements**.

   C. The selected elements will be replaced by a **single new shape** representing the new Element Group.

   D. Element Groups can only be made from elements on the same page in the call flow and the Element Group itself can only have one page. Page Entry or Page Connectors may not be included in an element group. All exit states into and out of the group are preserved.

4. **Configuring an Element Group**
   A. The contents of an element group can be viewed by double-clicking on the Element Group shape. The **Element Group Editor (EGE)** will take up the call flow editor area and the application's page tabs will temporarily disappear.

   B. To go back to the call flow editor, click the **Go Back** button at the bottom-right of the editor.

   C. Any element from the Elements pane can be dragged into the EGE except the Page Entry and Page Connector elements (as Element Groups only span a single page).

5. **Entry Points and Exit States**

   - Exit states that led to elements that are now part of the Element Group appear as **entry points** coming from the Group Start element within the EGE.

   - The destination of entry points can be remapped by configuring the exit states of the **Group Start** element.

### Math Element Settings

| Settings | |
|---|---|
| Type | The type of data to store the result. Possible values are: Element or Session. |
| Name | The name to assign to the data storing the result of the mathematical expression. This variable is **created** if it doesn't yet exist; or reassigned the new value if it does exist. |
| Expression | The mathematical expression to parse and evaluate. |

8. Examples:

A. Increment a variable that you've already defined in the application:

```
*Type        Session
*Name        errorCounter
*Expression  {Data.Session.errorCounter} + 1
```

B. Calculate Price including Sales Tax:

```
*Type        Session
*Name        totalPrice
*Expression  {Data.Element.dbRxinfo.rxprice} * {Data.Session.taxMultiplier}
```

**If you do not change the TYPE setting, then the result is stored into Element data (default) for the current Math element.**

## The Math Element (Folder: Elements/Math/)

1. The **Math** action element is used to evaluate mathematical expressions.
2. The mathematical expression is composed of operators and functions to be parsed and evaluated at runtime. Use parentheses to prioritize operations or for clarity.
3. Substitution variables may be included in the expression.
4. Boolean operators are fully supported. Boolean expressions are evaluated to be either 1.0 (true) or 0.0 (false).
5. The result is a **Double** value stored in either Element data or Session data specified by the user in the Settings tab.
6. The result of the expression is logged in the Activity Log with the name **Result.**
7. This list of supported operators and functions is from the ***Element Specifications for Cisco CVP*** reference manual.

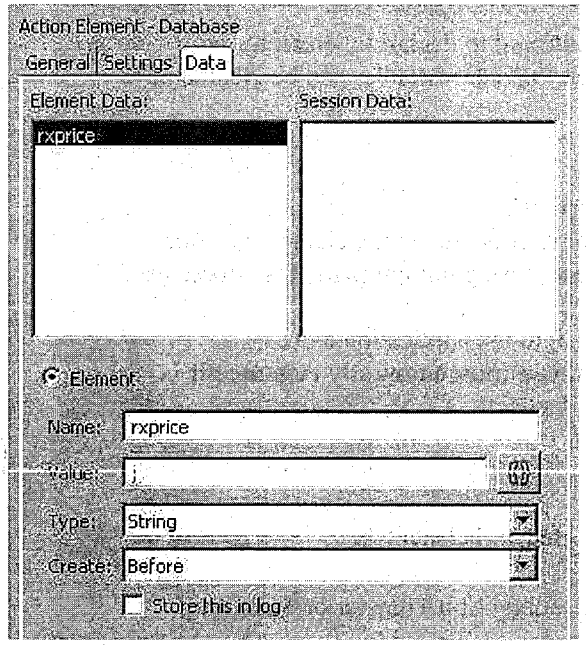
**Operators**

| Operation | Operator |
|---|---|
| Power | ^ |
| Boolean Not | ! |
| Unary Plus, Unary Minus | +x, -x |
| Modulus | % |
| Division | / |
| Multiplication | * |
| Addition, Subtraction | +, - |
| Less or Equal, More or Equal | <=, >= |
| Less Than, Greater Than | <, > |
| Not Equal, Equal | !=, == |
| Boolean And | && |
| Boolean Or | \|\| |

**Functions**

| Function | Syntax |
|---|---|
| Sine | sin(x) |
| Cosine | cos(x) |
| Tangent | tan(x) |
| Arc Sine | asin(x) |
| Arc Cosine | acos(x) |
| Arc Tangent | atan(x) |
| Arc Tangent (with 2 parameters) | atan2(y, x) |
| Hyperbolic Sine | sinh(x) |
| Hyperbolic Cosine | cosh(x) |
| Hyperbolic Tangent | tanh(x) |
| Inverse Hyperbolic Sine | asinh(x) |
| Inverse Hyperbolic Cosine | acosh(x) |
| Inverse Hyperbolic Tangent | atanh(x) |
| Natural Logarithm | ln(x) |
| Logarithm base 10 | log(x) |
| Exponential | exp(x) |
| Absolute Value / Magnitude | abs() |
| Modulus | mod() |
| Square Root | sqrt() |
| Sum | sum() |
| If | if() |

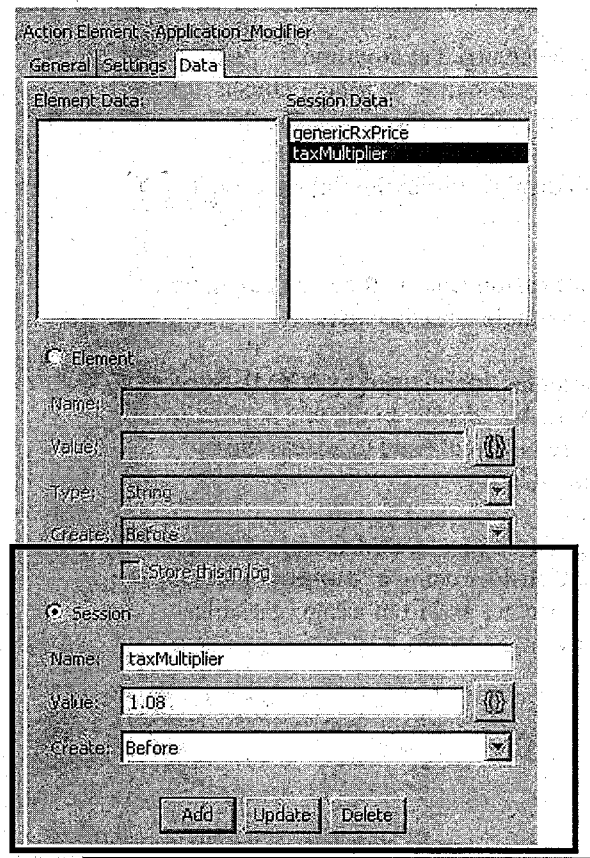
## Creating Data (or Reassigning a Value) in the Data Tab




**Element Data -** This is most often performed to initialize a variable that the element will re-assign at runtime (e.g., Database element)

1. In the Data tab, just below the Element Data box, select the radio button **Element**

2. Enter the following information:
   - **Name**: Enter a variable name, this will become element data for the current element.
   - **Value**: Enter any value or select a variable using Substitution.
   - Type: Suggestion of what this element contains. (Actually stored as String)
   - **Create**: When to perform the assignment. **Before** the current element executes or **After** the current element executes.
   - **Store in Log** - select this box if you'd like the variable name and value added to the Activity Log
   - Press **Add** at the bottom of the




**Session Data** - In the middle of the pane, select the radio button **Session**

1. Enter the following information:
   - **Name**: Enter a variable name
   - **Value**: Enter any value or select a variable using Substitution.
   - **Create**: When to perform the assignment. **Before** the current element executes or **After** the current element executes.

2. You **MUST** press ADD at the bottom of the pane for this to take effect.

3. If you change a variable definition you **MUST** press **UPDATE** at the bottom of the pane.

## Creating Variables

1. CVP Studio does not have a window or an element dedicated to declaring, creating, or assigning values to variables. Instead, Element and Session data variables can be created in the **Data tab** of almost any element.

2. **Element data**
   a. Created by an element (at runtime) or in its Data tab.
   b. It belongs to the element that creates it and no other element may change its value.
   c. May be used/treated as a Float, Integer, Boolean, String but internally it is stored as a String.
   d. Deleted by VXML Server when the caller's visit to the application ends.
   e. **Through CVP 7, if you rename the element, you must manually rename all references to its element data.**

3. **Session data**
   a. Created in many ways:
      - By an element (at runtime) or in an element's Data tab;
      - Passed in from ICM
      - Passed in from another Studio application (App Transfer or Subdialog Invoke)
      - Custom Java component
   b. It belongs to the application, so its value may be changed at any time.
   c. When created through custom Java, a Session variable may be a Java object.
   d. Deleted by VXML Server when the caller's visit to the application ends.

4. Session and Element variables defined in the Data tab display within the Substitution Tag Builder.

5. When creating and/or assigning a value to Element or Session Data in the Data tab, there is no difference to the system if the Element or Session variable already exists.

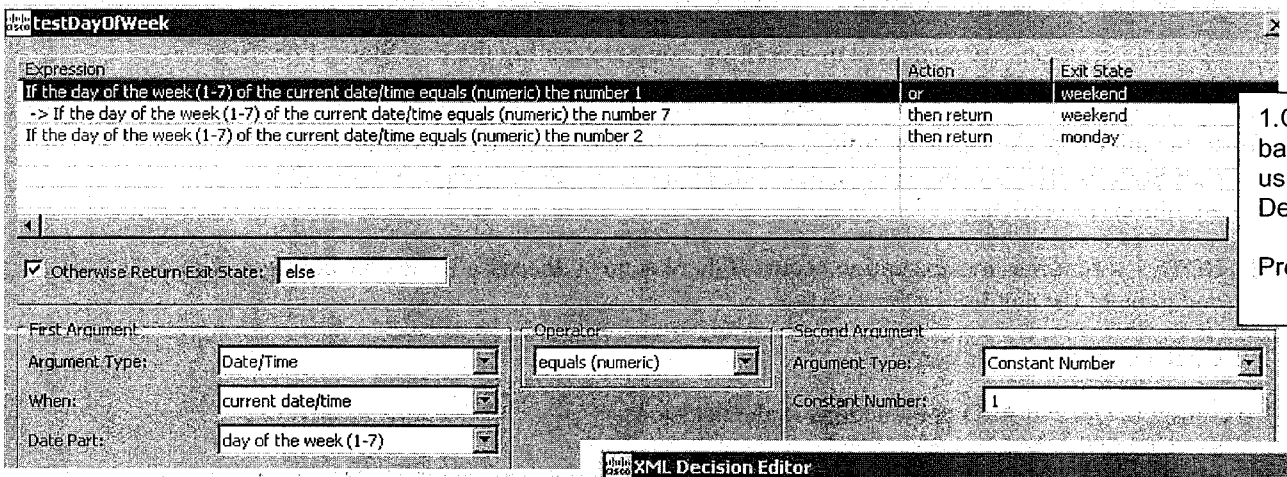
6. **Application Data** and **Global Data** are persistent variables, maintained by VXML Server between phone calls. App and Global data are read/write by all calls to the one application, or to all applications, respectively. Custom Java is required to create and to access these variables. They do not display anywhere in the Studio GUI.

7. **Best Practice:** To make your variable assignments easier to locate, **add Comments** to specify which elements create data in their Data tabs. Consider using a **'dummy'** element such as **ApplicationModifier** named appropriately to declare and to re-assign variables (**dummySetVars**).

## Modifying the XML of a Decision Element (Reference: CVP Studio Users Guide)

After you've configured a decision within the Decision Editor you can manually edit the XML created by the Decision Editor to perform operations such as:

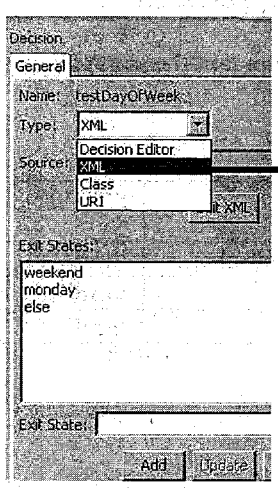
- Copy/paste - to create multiple decisions similar to one already created
- Cut/paste - to change the order of evaluation of the decisions

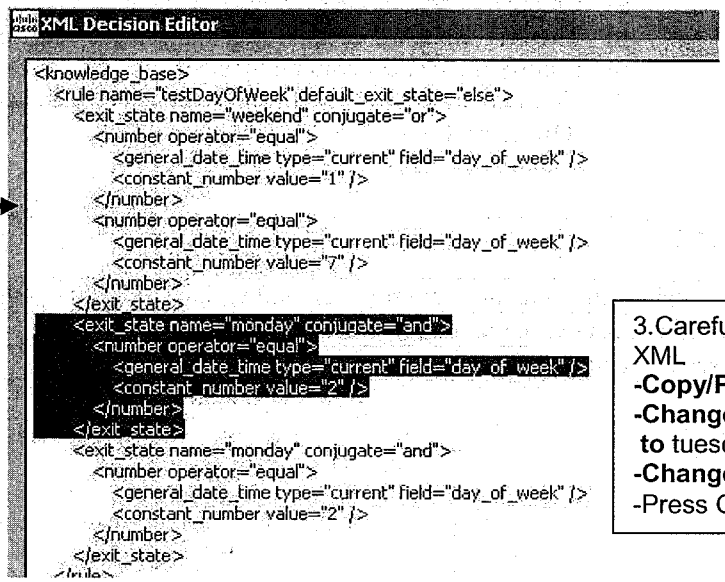



1.Create the basic decision using the Decision Editor.

Press OK

2.Open the XML Decision Editor

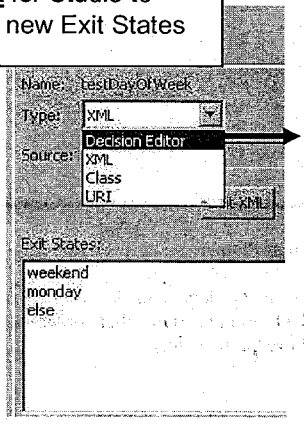


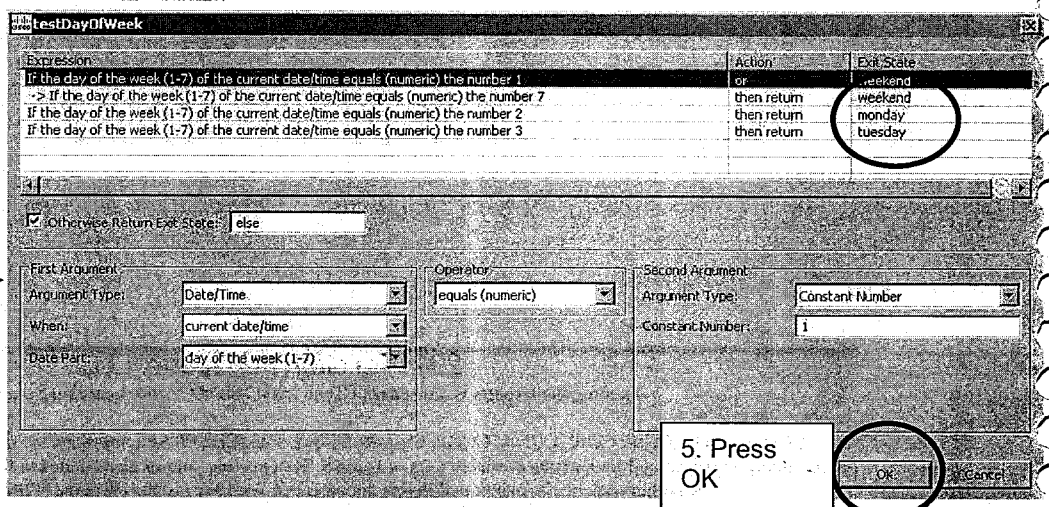




3.Carefully edit the XML
**-Copy/Paste**
**-Change** monday **to** tuesday
**-Change** 2 **to** 3
-Press OK

4. You **MUST** re-open the Decision Editor

-Press **OK** for Studio to create the new Exit States




5. Press OK

D. Finally, enter the name of an **Exit State** to follow if this condition is satisfied. At the **top/right of the Decision Editor**, click the Exit State column next to the expression you just configured and enter the name of an exit state to execute when this statement is satisfied. Call Studio will create this exit state in the workspace.

E. To add another decision, **right click and select "Add Another Exit State"**

F. Multiple decisions are evaluated in order (top-down) executing the Exit State of the first decision that is satisfied.

G. In the middle of the Decision Editor, make sure that the checkbox for **Otherwise Return Exit State** is selected and enter an exit state name to execute when none of the above conditions are satisfied.

H. **NOTE** – a runtime Java exception occurs if there is no "Otherwise" exit state configured and none of the conditions are satisfied.

I. Click **OK** to save your work.

6. To configure **OR** and **AND** conjunctions in the decisions.
   a) Create the first expression and exit state.
   b) Then right-click the expression and select **'Add New Expression'**
   c) Modify the **Action** column by selecting **'and'** or **'or'**
   d) You may continue to add expressions but they must contain the same conjunction (OR or AND).

| Expression | Action | Exit State |
|---|---|---|
| If element data from element 'ctNumErrors' and variable 'count' >= the number 3 | **'and'** | callMon |
| → >> If session data variable 'dayOfWeek' equals 'Saturday' | **'and'** | callMon |
| → >>If session data variable 'vip' is 'false' | **'and'** | callMon |

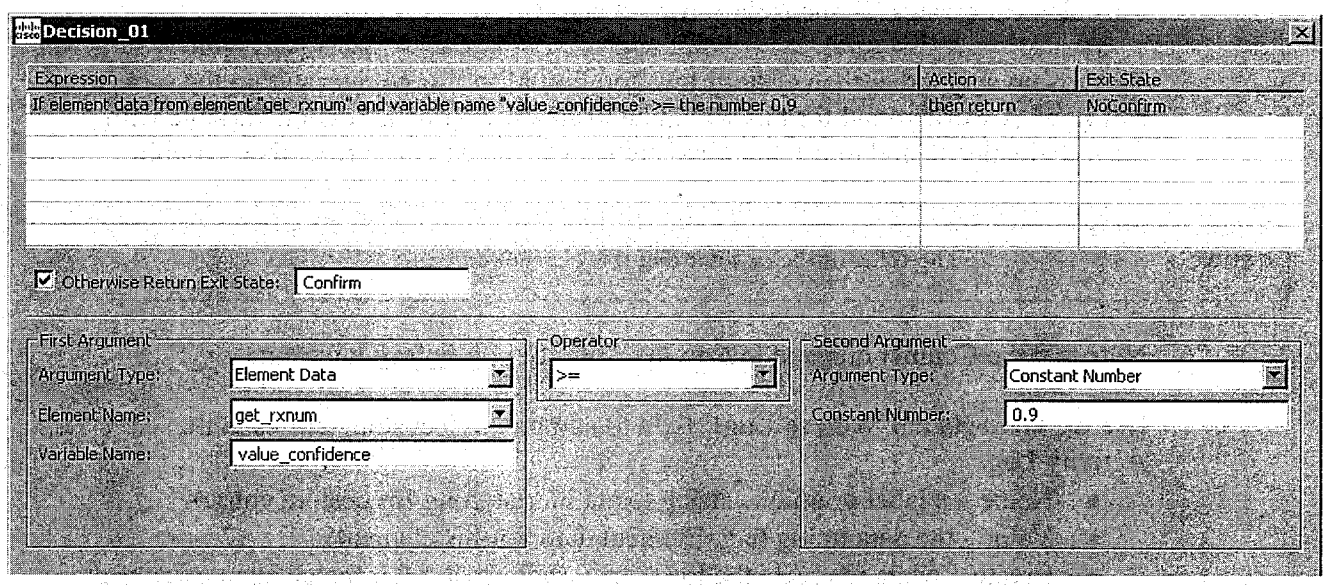
## The Decision Element

1. The **Decision element** is used to branch code based on one or more conditions and must be configured with at least 2 exit states, but may have more.

2. The decisions specified are evaluated top-down until one is satisfied, at which time the callflow exits down the associated exit state, no other conditions are evaluated. An **Otherwise Exit State** should always be specified.

3. The easiest method of configuring the Decision element is to use the **Graphical Decision Editor**. Although quick copy/paste functionality can then be accessed by switching to the XML.

4. Select **Decision Editor** from the **Name**: menu then press the **Use Decision Editor** button.




5. Steps to configuring a decision in the Decision Editor
   A. In the **First Argument** box at the bottom, select:

   - **Argument Type**: Type of data (Element, Session, CallData, etc.).
   - **Element Name**: Select an element
   - **Variable Name**: Select or enter the variable name

   B. In the **Operator** box at the bottom, select an operator:

   - **Number comparisons** – greater, greater_equal, less, less_equal, equal, not equal
   - **String comparisons** – contains, starts_with, ends_with, equals, and their negatives
   - **Boolean comparisons:** True, False, exists, does_not_exist

   C. If the **Second Argument** box appears (it disappears for Boolean Comparisons) select the **Argument type** and then the required information.

## Counter Element (Folder: Elements/Math/)

1. The **Counter** element is used to create and increment a counter named **count** which is stored as element data. A typical use for the Counter element would be in a loop in the call flow that increments the count until a decision element decides that the loop must end.

2. Revisiting a Counter element within a phone call will automatically update the count.

3. The **initial value** of count is defined as a configuration setting. In addition, the element may be configured to **increment or decrement** with a **user defined step size.**

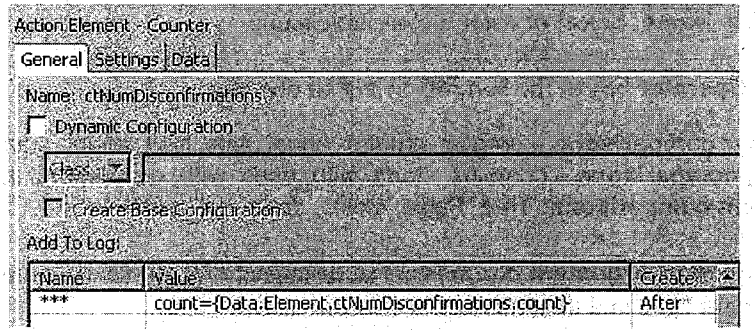
4. This element creates **Element data** named **count.**

5. There is no way, other than custom Java to reinitialize this counter within the same call flow. A workaround for this would be to create your own Session variable with the initial value 0, and to use the Math element to increment or reset that variable.

| Element Data | |
|---|---|
| `count` | The value of the counter that this element creates. |

6. Note that the Counter element logs to the Activity Log **before** it executes.

7. Use the **General** tab **'Add to Log:'** field to add up to 10 custom entries to the Activity Log.
   - Name – a label or unique string to aid in searching for custom entries.
   - Value - the data to log (e.g. Element data for this element)
   - Create – when to create the log entry – Before or After the element executes. For the Counter element, select **AFTER**
   - **NOTE - When adding multiple entries to the Log within one element, each must have a distinct entry in the 'Name' field for all to display.**



# Chapter 6

# Counters, Decisions, Variables, Math Element, Element Groups

- Counter Element
- Decision Element
- Creating Variables
- Math Element
- Element Groups

# APLICATION MODIFIER = DECLARE VARIABLES

SESSION = GLOBAL

ELEMENT = LOCAL (NO ELEMENT)

maxNBest = max # results to get back from ASR

1234 ⟹

ASR = recognizes

| N-Best List | confidence |
|---|---|
| 1234 | .6 |
| 1224 | .4 |
| 1204 | .2 |
| 1334 | .01 |

CLEARS AS MUD

◇ = DECISION ELEMENT EXEC ON VXML SERVER
HAS >1 EXIT STATE

□ = VOICE ELEMENT - CREATS VOICEXML

AUDIO
DIGITS
NUMBER
CURRENCY
FORM
MENU

} CREATE ELEMENT DATA VALUE
VALUE_CONFIDENCE
(ETC)

□ = ACTING ELEMENT = EXECUTES ON VXMLSERVER
COUNTER ELEMEN - CREATES ELEMEN DATA . COUNT
MATH - DO ANY MATH → (YOU DECIDE)

b) Now click on the first **Expression** line of the Decision Element, and in the **Exit State column**, and enter the name of the exit state to follow when that condition is satisfied: **mmdd**

**c)** Right click on the first **Expression** line of the Decision Element, and select **Add A New Exit State.**

d) Now, click on the Second **Expression** line of the Decision Element, and enter the information below to check that the spoken input does not contain an invalid "?"

- First Argument:
  Argument Type: **Element Data**
  Element Name: **getDepDate**
  Variable Name: (type in) **value**
- Operator: **Does Not Contain**
- Second Argument:
  Argument Type: **Constant String**
  String: **?**

e) Click on the second **Expression** line of the Decision Element and in the **Exit State** column enter the name of the exit state to follow when that condition is satisfied: **yyyymmdd**

f) In the **middle** of the Decision Element, select the checkbox **Otherwise Return Exit State**: and also enter the name of an exit state to follow if neither of the above is satisfied: **invalid**

g) Press **OK**.

4. Connect the **testDepDate→mmdd** exit state to a **pItineraryMMDD** audio element that speaks the itinerary including the **getDepDate.value** as Say it Smart Date in MMDD format.

5. Connect the **testDepDate→yyyymmdd** exit state to a **pItineraryYYYYMMDD** audio element that speaks the itinerary including the **getDepDate.value** as Say it Smart Date YYYYMMDD (You can copy/paste the pItineraryMMDD you just created and modify it).

6. Connect the **testDepDate→invalid** exit state to a **pInvalid** audio element that tells the caller to speak the date including the month, day and year. Then, loop back to the **getDepDate** element.

[End of Chapter 5]

7. Connect the **getDepCity→done** exit state to an Audio element (named **pItinerary**) to speak the caller's selected itinerary.
   - ***You've selected the*** {Data.Element.getAdventure.value } ***adventure departing from*** {Data.Element.getDepCity.value}.

8. Connect **pItinerary** to an Audio element that says '*I will pass this information along to the next available travel agent who will take your payment information*'

9. Connect to a new CVP Subdialog Return element where you return:

| | |
|---|---|
| CallerInput: | transfer |
| External VXML0: | TripType= {Data.Element.getAdventure.value} |
| External VXML1: | DepCity={Data.Element.getDepCity.value} |

10. Connect all **max_noinput**, **max_nomatch**, and **max_disconfirmed** paths to an audio element (named **pCSR**) that tells the caller to '*hold for an agent*'. Then connect to a CVP_Subdialog_Return element with the Caller Input setting **'transfer'**

**Challenge (Only if you are done early)**

1. Before the **pItinerary** element, create a Form element **getDepDate** to collect the departure date (do **NOT** use Form with Confirm).

2. Prompt the caller '*Tell me the full departure date including the year. Or you may key it in as month month day day*'
   - Voice Grammar: **builtin:grammar/date**
   - Dtmf Grammar: **builtin:dtmf/digits?length=4**
   - Set the Confidence to 0

3. **(The steps for this follow this overview)**

   After executing the Form element, use a **Decision** element (**testDepDate**) to check if
   a) If **nbestInputmode1** is **dtmf**
   b) If the collected spoken date (**value**) contains only valid characters (doesn't contain ?)
   c) else, exit down the invalid path.

   a) At the bottom of the Decision element, use the pull-down menu to specify:
   - First Argument:
     Argument Type: **Element Data**
     Element Name: **getDepDate**
     Variable Name: (type in) **nbestInputmode1** (upper-case **I**)
   - Operator: **equals (String)**
   - Second Argument:
     Argument Type: **Constant String**
     String: **dtmf**

## Chapter 5 Programming Lab

Create a travel application that collects the caller's adventure type and departure city.

1. Before you begin to write the Studio application, copy the folder named 'grammars' from the directory **C:/CVP7CourseFiles/grammars** to **C:/Inetpub/wwwroot/grammars**

2. Create a new CVP Project, named **MyApp.** Set the Project / Properties / General Settings with the correct Gateway Adapter.

3. Use a CVP Subdialog Start element

4. Use an Audio element (named **pWelcome**) to say *Welcome to Travel Time where every trip is an adventure.*

5. Use a Form element (named **getAdventure**) to create an inline menu:

Audio

- Initial: ***Which adventure would you like? safari, skiing, or rafting?***
- End > Done*:* ***You've selected*** **{Data.Element.getAdventure.value}**

Settings (**Warning!!** don't use the **Voice Grammar** setting!)

- Voice Keyword: safari
- Voice Keyword: skiing
- Voice Keyword: rafting
- DTMF Keypress: 1[safari]
- DTMF Keypress: 2[skiing]
- DTMF Keypress: 3[rafting]

6. Use a Form (named **getDepCity**) to ask the caller where they would like to depart from.

Settings:

- **Voice Grammar**: **http://10.1.78.xx/grammars/city.xml** (insert the IP address of your pc here)
- **DTMF Grammar**: **http://10.1.78.xx/grammars/citydtmf.xml** (insert the IP address of your pc here)
- **Field Slot**: **travel_city** (required by the grammar)

NOTE: The grammars allow the caller to **say Boston or press 1, Say San Francisco or press 2. Say Raleigh or 3.**

Audio:

- **Form Initial**: *From which city would you like to depart? Say Boston or press 1. San Francisco or press 2. Say Raleigh or press 3.*
- **End >> Done**: *Leaving from {Data.Element.getDepCity.value}*

## Chapter 5 Review Questions

1. Which settings do you configure when working with an 'external' grammar that resides on the Media Server?

2. Which setting do you configure to use the built in grammar named digits? And what would you specify for that setting to collect 5 digits?

3. How do you configure a Form element to allow the caller to say Savings or My Savings Account or press DTMF 1 and have all of these return 'savings' into the value variable?

4. What are two variables that the Form element creates, but no other collection elements create?

5. If you are requesting multiple results from the ASR recognizer using the Form element, which Element data variable holds:

    - The number of results returned?
    - The best guess (interpretation) of what the caller entered?
    - The second best guess?
    - The confidence level of each interpretation?

## Cisco DTMF-Only Gateway Adapter

A. N_Option_Menu elements – do not fill in the **Option X Voice** settings (where X is 2 - 10 as applicable) settings.

B. Form and Form_With_Confirm elements:

- **Voice Keyword** and **Voice Grammar** must not be configured.
- **DTMF Grammar** may contain builtin:dtmf/<grammarName> but may not point to external grammars using http.
- **DTMF Keypress** – Return values in square brackets are ignored, only the DTMF input is returned into the value variable. To allow callers to press '*' key, you must use *

C. With DTMF input recognition, the voice browser terminates the input process as soon as it detects a valid input. As a result, the inline DTMF grammar must not contain two inputs such that one is a substring of the other. Otherwise, entering the longer input will trigger recognition of the shorter input. For example, if DTMF 1 and 12 are both configured for **DTMF Keypress**, a DTMF input 12 will be recognized as 1 by the voice browser.

D. Hotlink element:
- The **DTMF** setting must contain only inline DTMF. To allow callers to enter a star key, use backslash then star (*).

## Form Element Debugging Tips

If you receive the **error.noresource (or com.cisco.media.resource.failure.asr )** it does not mean the ASR resource is down, it usually indicates one of the following:

A. The ASR resource couldn't find a grammar you listed in the "Voice Grammar" or "DTMF Grammar" setting.

   i. Perhaps you meant to create an inline grammar and should have listed the utterance in the Voice Keyword or DTMF Keypress setting.

   ii. If you do indicate an external grammar, ensure that it is retrievable by entering the URI into a web browser.

B. The inline "DTMF Keypress" contains invalid characters. It must contain only digits (optionally followed by square brackets containing an interpretation).

   i. **Invalid: pharmacy [refills]** or **one [refills]**
   ii. **Valid: 1[refills]** (unless using the DTMF-only gateway adapter)

C. The inline "Voice Keyword" is not in a valid format. The utterance may not contain any punctuation.

   i. **Invalid**: I don't know [agent]
   ii. **Valid**: I do not know [agent]

D. The "Help Keyword" and "Help Keypress" may NOT contain an interpretation in square brackets, as it is already being interpreted as 'help' by being part of this setting.

   i. **Invalid**: I need help [help]
   ii. **Valid**: I need help

## Form Element to Invoke Builtin Grammars

1. The Builtin grammars (*digits, number, currency, date, time, phone, boolean*) are pre-compiled grammars **provided with every VoiceXML Gateway and ASR server.**

2. They can be explicitly referenced using the Voice Grammar and DTMF Grammar setting. This allows you to:
   - Use the Form element to create the **collect_noinput_count** and **collect_nomatch_count** variables while collecting standard DTMF input like digits, number, currency
   - **Combine different grammars**. For example, to collect a date from the caller using DTMF, the builtin date grammar requires YYYYMMDD input format. So, you might want to pair the date voice grammar with the digits DTMF grammar.
     **Example,** ***Please tell me your birthday, or key it in as month, day, year***
   - Inline grammars can also be combined with external grammars, and inline grammars discussed earlier.

3. After collecting the input, the Element variable **nbestinputmode1** will equal the string (all lower case) **dtmf** or **voice** indicating the caller's input mode.

## Form Element Settings for Builtin Grammars

1. **Voice Grammar** – The URI of a builtin delivered with the ASR Server:
   **builtin:grammar/digits**
   **builtin:grammar/number**
   **builtin:grammar/currency**
   **builtin:grammar/date**
   **builtin:grammar/time**
   **builtin:grammar/phone**
   **builtin:grammar/boolean**
   - The builtin digits grammar also accepts the following formats to specify the number of digits to collect: **builtin:grammar/digits**
     **builtin:grammar/digits?minlength=4**
     **builtin:grammar/digits?maxlength=6**
     **builtin:grammar/digits?length=4**

2. **DTMF Grammar** – URI of a builtin DTMF grammar on the gateway and ASR Servers:
   **builtin:dtmf/date**
   **builtin:dtmf/time**
   **builtin:dtmf/digits**
   **builtin:dtmf/phone**
   **builtin:dtmf/number**
   **builtin:dtmf/boolean**
   **builtin:dtmf/currency**

   - The builtin **digits** grammar also accepts the following formats to specify the number of digits to collect: **builtin:dtmf/digits**
     **builtin:dtmf/digits?minlength=4**
     **builtin:dtmf/digits?maxlength=6**
     **builtin:dtmf/digits?length=4**

6. **DTMF Grammar** – URI of an external DTMF grammar that resides in a file or on a web server. You may indicate a custom grammar or a builtin grammar. This setting is repeatable and optional.

   - **External grammars** are invoked using the http:// or file:// notation. Note that all grammars must be retrievable by the ASR server. When using custom grammars, DTMF grammars must be specified to allow touchtone entries.

     E) For example,

**http://10.1.78.12/citydtmf.xml**

```
<?xml version="1.0" " encoding="UTF-8"?>
<grammar xml:lang="en-us" xmlns="http://www.w3.org/2001/06/grammar" mode="dtmf" type="application/grammar+xml"
    version="1.0" root="ROOTVOICE">          <!- - Voice Grammar for OSR 3 - - >
    <rule id="ROOTVOICE" scope="public">
     <one-of>
      <item> 1  <tag>travel_city='Boston';</tag>  </item>
      <item> 2  <tag>travel_city='San Francisco';</tag>  </item>
      <item> 3  <tag>travel_city='Raleigh';</tag>  </item>
     </one-of>
    </rule>
   </grammar>
```

7. If you are using the **Cisco DTMF-Only Gateway Adapter**:
   - Set Input Mode: **dtmf**
   - Do **NOT** fill DTMF Grammar
   - Fill in DTMF Keypress (repeatable setting) but do **NOT** use the square brackets to assign an interpretation to a DTMF Keypress

5. For a grammar that returns multiple slots per utterance, there are two ways to configure the Form element to store the slot/value pair information:

   A. Leave the **Slot Element Data** setting empty. The Form element will create element data named **nbestInterpretationX** (where X is from 1 to the length of the n-best list) in the format: "+Slotname:value+Slotname:value...". A developer would then need to parse this.

   I. For example, if the caller says Boston, Massachusetts, then the slots returned are 'city' and 'state'. And the element data created would be nbestInterpretation1:
   {Data.Element.Form01.nbestInterpretation1} +city:Boston+state:MA

   II. Note that if **MaxNbest > 1** , then many element data variables might be created:
   {Data.Element.Form01.nbestInterpretation1} +city:Boston+state:MA
   {Data.Element.Form01.nbestInterpretation2} +city:Austin+state:TX
   {Data.Element.Form01.nbestInterpretation3} +city:Mosstown+state:FL

   B. Configure the repeatable setting **Slot Element Data** setting with the names for all the slots that can be returned. Then Element data with these slot names, followed by an integer (1, 2, 3, etc) will be created with the corresponding values.

   I. For example, if slot_element_data contains 'city', 'state', and 'zip' then the element data variables created might be: "city1", "city2", "city3", "state1", "state2", "state3","zip1", "zip2" and "zip3"

   For example: {Data.Element.Form01.city1} boston -and-
   {Data.Element.Form01.state1} MA -and-
   {Data.Element.Form01.zip1} 02210

   II. Note that if **MaxNbest > 1** , then many element data variables might be created:
   {Data.Element.Form01.city1}, {Data.Element.Form01.city2},{Data.Element.Form01.city3}
   {Data.Element.Form01.state1}, {Data.Element.Form01.state2},{Data.Element.Form01.state3}
   {Data.Element.Form01.zip1}, {Data.Element.Form01.zip2},{Data.Element.Form01.zip3}

```
<?xml version="1.0" " encoding="UTF-8"?>
<grammar xml:lang="en-us" xmlns="http://www.w3.org/2001/06/grammar" mode="voice" type="application/grammar+xml"
    version="1.0" root="ROOTVOICE">          <!- - Voice Grammar for OSR 3 - - >
    <rule id="ROOTVOICE" scope="public">
     <one-of>
      <item> Boston          <tag> city='Boston'; state='MA';zip='02210';</tag>  </item>
      <item> Bean Town       <tag> city='Boston'; state='MA';zip='02210';</tag>  </item>
      <item> San Francisco   <tag> city='San Francisco'; state='CA';zip='95010';</tag>  </item>
      <item> Raleigh         <tag> city='Raleigh'; state='NC';zip='27954';</tag>  </item>
      <item> Austin          <tag> city='Austin'; state='TX';zip='78701';</tag>  </item>
      <item> Mosstown        <tag> city='Mosstown'; state='FL';zip='33134';</tag>  </item>

     </one-of>
    </rule>
```

## Form Element: External Grammars (For ASR Gateways Only)

1. External grammars are custom grammars that reside in a file. They can be voice grammars and/or dtmf grammars.

2. If you collect touchtones from the caller then you **must** specify a dtmf grammar (external, builtin, or inline).

3. **Voice Grammar** – The URI of an external voice grammar that resides in a file or on a web server. The grammar must be retrievable by the ASR system.

   **http://10.1.78.12/city.xml**

   - This setting is repeatable to allow you to specify multiple grammars to recognize against. It is also optional.

   - **External custom grammars** are invoked using the http:// or file:// notation. Note that all grammars must be retrievable by the ASR server. For example,

**http://10.1.78.12/city.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xml:lang="en-us" xmlns="http://www.w3.org/2001/06/grammar" mode="voice" type="application/grammar+xml"
    version="1.0" root="ROOTVOICE">        <!- - Voice Grammar for OSR 3 - - >
    <rule id="ROOTVOICE" scope="public">
      <one-of>
        <item>  Boston          <tag>travel_city='Boston';</tag>   </item>
        <item>  Bean Town       <tag>travel_city='Boston';</tag>   </item>
        <item>  San Francisco   <tag>travel_city='San Francisco';</tag>  </item>
        <item>  Raleigh         <tag>travel_city='Raleigh';</tag>   </item>

      </one-of>     utterance          Field slot  =  value
    </rule>
  </grammar>
```

4. In the grammar, if the <tag> contains a slot of the form ***slotName=slotValue*** (example, travel_city='Boston'), then the ***slotValue*** will only be copied into the Element Data named 'value' if the ***slotName*** is specified in the Form element's **Field Slot** (or Field Name) setting.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml" version="2.1"
application="/CVP/Server?audium_root=true&amp;calling_into=MyApp" xml:lang="en-US">
    . . .
  <form id="audium_start_form">
    . . .
 </form>
 <form id="start">
  <property name="inputmodes" value="dtmf voice" />
  <block>
    . . .
  </block>
  <field name="foundation_fld" modal="false">
   <property name="inputmodes" value="dtmf voice" />
   <grammar mode="voice" type="application/grammar+xml" version="1.0" xml:lang="en-US"
            root="ROOTVOICE">
    <rule id="ROOTVOICE" scope="public">
     <one-of>
      <item>
       ohio
       <tag>foundation_fld='Ohio';</tag>
      </item>
      <item>
       the buck eye state
       <tag>foundation_fld='Ohio';</tag>
      </item>
      <item>
       iowa
       <tag>foundation_fld='Iowa';</tag>
      </item>
      <item>
       utah
       <tag>foundation_fld='Utah';</tag>
      </item>
     </one-of>
    </rule>
   </grammar>

   <grammar mode="dtmf" type="application/grammar+xml" version="1.0" xml:lang="en-US"
    root="ROOTDTMF">
    <rule id="ROOTDTMF" scope="public">
     <one-of>
      <item>
       1
       <tag>foundation_fld='ohio';</tag>
      </item>
      <item>
       2
       <tag>foundation_fld='iowa';</tag>
      </item>
      <item>
       3
       <tag>foundation_fld='utah';</tag>
      </item>
     </one-of>
    </rule>
   </grammar>
```

A portion of a debug log is shown below.

```
1.  ------------------------------------
2.  ------- New Application Visit ------
3.  ------------------------------------
4.  Java Application Server SessionID = 84F85FE2740D0D6B9679D5175203135
5.  Cisco CVP VXML Server SessionID=10.1.78.20.1143043731445.2.MyApp
6.  ------------------------------------
7.  --------- New Page Request ---------
8.  ------------------------------------
9.  ------- HTTP Request Headers -------
10. Header Name = "host"  Header Value = "10.1.78.20:8080"
11. Header Name = "content-type"  Header Value = "application/x-www-form-urlencoded"
12. Header Name = "connection"  Header Value = "close"
13. Header Name = "accept"  Header Value = "text/vxml, text/x-vxml, application/vxml, application/x-vxml,
    application/voicexml, application/x-voicexml, text/plain, text/html, audio/basic, audio/wav, multipart/form-data,
    application/octet-stream"
14. Header Name="user-agent" Header Value="Cisco-IOS-C3725/12.3(11)T6 VoiceXML/1.0"

15. ------ Request HTTP Arguments ------
16. Parameter Name = "audium_application"  Parameter Value #0 = "MyApp"
17. -------------- Timing --------------
18. Request Received on:  Wed Mar 22 11:08:51 EST 2006
19. Response Returned on: Wed Mar 22 11:08:51 EST 2006
20. --------- VoiceXML Response --------
21. <?xml version="1.0" encoding="UTF-8"?>
22. <vxml version="2.0" application="/CVP/Server?audium_root=true&amp;calling_into=MyApp">
23.  <form id="audium_start_form">
24.    <block>
25.       <assign name="audium_vxmlLog" expr="'''' />
26.       <assign name="audium_element_start_time_millisecs"  expr="new Date().getTime()" />
27.       <goto next="#start" />
28.    </block>
29.  </form>
30.
31.  <form id="start">
32.    <block>
33.       <prompt bargein="true"> welcome to CVP pharmacy.</prompt>
34.       <assign name="audium_vxmlLog" expr="audium_vxmlLog" />
35.       <submit next="/CVP/Server" method="post" namelist="audium_vxmlLog"/>
36.    </block>
37.   </form>
38. </vxml>
```

## Enabling Detailed VoiceXML Debug Logs on CVP VXML Server

A. There is a debug option that can be enabled to log **all** interactions between the Voice Gateway and VXML Server, including VoiceXML, http headers, and custom inline grammars created by the Form element..

B. **WARNING**: When this feature is enabled, CVP VXML Server will produce very large log files very rapidly; it is NOT recommended that this option be enabled on production systems unless necessary.

C. Note that one debug log will be created for each phone call, the names will be similar to:
**10D32187D908B02569B4EA4EF54F7044.txt**

D. Enable VXML Debug Logs as follows:
- In Call Studio, click on the project name in the Navigator pane
- Select **Project / Properties / Call Studio / General Settings**
  - A. Next to the Loggers window, press the button **Add..**
  - B. Enter any name you choose for the directory that will store the logs
    Enter the following class name (case sensitive)
  - **C. OK**

| |
|---|
| Name: **VxmlLogs**<br>Class: **com.audium.logger.application.debug.ApplicationDebugLogger** |

- Save, deploy and update the application. Place a call.
- One log file for each phone call will be stored into the folder
**C:Cisco/CVP/VxmlServer/applications/<appname>/logs/MyDebugLogs/**

E. Debug logs are useful during development to gain a better understanding of
- The VXML created by VXML Server and
- The data returned by the VoiceXML Gateway to the VXML Server based upon the caller's response
- View the grammar created by a Form element.

F. A portion of the debug log is shown on the next page.

| **Multiple Slots from the Grammar**<br><br>**If a grammar returns multiple slots, enter each slot name into the Setting named Slot Element Data. This causes the system to create Element data variables with the name of each slot returned (followed by 1, 2, 3, etc)** | |
|---|---|
| **<*FieldSlotName*>1, 2, 3** | If you enter the slot name in the Setting 'Slot Element Data' then you will have Element data with the slot name followed by 1, 2, 3 (eg: slot **address** → element data **address1**, **address2**, etc. |

C) Do **NOT** follow the Help keypress with square brackets [ ] or an error.semantic event will occur at runtime

D) NOTE – In a Form with Confirm element, the Help Keyword and Help Keypress are **only active** during the collection stage, NOT during the confirmation.

| Setting | Examples: |
|---|---|
| + Help DTMF Keypress | 911 |

6. Forms create Element Data listed below. Forms also create two unique variables **collect_noinput_count** and **collect_nomatch_count** that contain the number of noinput and nomatch events which occurred during the collection of caller input.

| Form Element Data | |
|---|---|
| **value** | The interpretation of the caller's input as specified by the grammar. |
| **value_confidence** | This is the confidence value of the captured input. |
| **confirm_confidence** (Form with Confirm Element) | This is the confidence value of the captured confirm utterance. |
| **nbestInputmode1** | The input mode 'voice' or 'dtmf' |
| **nbestUtterance1** | The utterance from the caller as it matched the grammar.<br>Example, `from Boston to Raleigh` |
| **bestConfidence1** | The confidence value returned by the recognizer. Float between 0.0 – 1.0 (DTMF is 1.0) |
| **nbestInterpretation1** | The interpretation of the caller's input as returned by the recognizer. This will be in the form:<br>`+slotA:valueA+slotB:valueB+slotC:valueC+`<br>Example, `+src:Boston+dest=Raleigh+` |
| **collect_noinput_count** | The number of NoInput events during the collection phase. |
| **collect_nomatch_count** | The number of NoMatch events during the collection phase. |
| **Multiple Results from the Recognizer** (the maxNbest setting must be configured greater than 1) | |
| **nbestLength** | The number of results returned from the recognizer when the maxNbest setting is changed to a number greater than 1. |
| **nbestInputmode1,2,3,...** | The input mode 'voice' or 'dtmf' for each result returned from the recognizer |
| **nbestUtterance1,2,3,...** | The utterances returned from the recognizer |
| **bestConfidence1,2,3,...** | The confidence values returned by the recognizer |
| **nbestInterpretation1,2,3,...** | The interpretations of the caller's input as returned by the recognizer. |

3. **DTMF Keypress** – This repeatable setting allows you to create a list of alternative key presses the caller can enter. Specify one **sequence of key presses** per DTMF Keypress.

   A) By default, the touchtones will be returned into the element data named '**value**'.

   B) To configure a different **interpretation** for the return value, follow the keypress by square brackets [ ] containing the interpretation value. This is useful to provide the same return value as used for the voice recognition.

   C) Any key presses entered by the caller that are not in this list will cause a NoMatch event to occur.

   **D)** The **DMTF Keypress** may ONLY contain characters (0-9, *, #). But **Key presses may consist of more than one digit**

| Setting | Examples: |
|---|---|
| + DTMF Keypress | 1 [Ninth Street] |
| + DTMF Keypress | 2 [Massachusetts Avenue] |
| + DTMF Keypress | ** [Agent] |

   E) **WARNING:** If you select the **Cisco-DTMF Gateway** adapter, do not use the square brackets as it results in an 'error.semantic' event occurring at runtime with some CVP versions.

| Setting | Examples: |
|---|---|
| + DTMF Keypress | 1 |
| + DTMF Keypress | 2 |
| | |

4. **Help Voice Keyword** – (Optional) You can indicate an inline voice utterance to activate the "Help" audio prompt being played.

   A) If no Help audio is provided, the NoMatch audio group is played. Otherwise, the Initial audio group is played.

   B) Do **NOT** follow the utterance with square brackets [ ] or an error.semantic event will occur at runtime

| Setting | Examples: |
|---|---|
| + Help Voice Keyword | Help me please |
| + Help Voice Keyword | I do not know |

5. **Help DTMF Keypress** – Indicate one or more DTMF key presses to activate the Help audio group. If no Help audio is provided, the NoMatch audio group is played. Otherwise, the Initial audio group is played. Format of the DTMF Keypress is one touchtone (0-9, #, *)

## Form Element to Create a Menu

1. You can use the Form element to create a **menu** for the caller. The benefit of a Form element for a menu is that you can continually add more options in the Settings tab, and the DTMF entry can be more than one touchtone in length.

2. **Voice Keyword** – This repeatable setting allows you to create a list of alternatives the caller can say.
   A) **Voice Keyword** indicates an **utterance** the caller may say. By default, the same words will be returned into the element data named '**value**'.

   B) To configure a different **interpretation** for the return **value**, follow the utterance by square brackets **[ ]** containing the interpretation value. This is useful to provide synonym utterances that return the same value into the variable.

   C) The **utterance** may NOT contain punctuation or abbreviations, but the interpretation may.

| **Setting** | **Examples:** | |
|---|---|---|
| +Voice Keyword | Ninth Street | Return this utterance into the 'value' variable |
| +Voice Keyword | Ninth [Ninth Street] | Caller can say 'Ninth' but return 'Ninth Street' into the 'value' variable |

   D) You may optionally assign the speech recognition language to use for a voice keyword when mixing languages within a menu, use the syntax:
   **contextLang;itemLang;weight;utterance[interpretation]**

- **contextLang**: (optional) Include this item in the grammar only if the Default Language matches this.
- **itemLang**: (optional) The language context to use for recognizing this caller utterance. This sets the xml:lang attribute for this item. If omitted none is included.
- **Weight**: (optional) The probabilistic weight for this item as compared to other items.
- **Utterance**: (required) What the caller may say
- **[Interpretation]** : (optional) The value to assign to the value variable

**Note**, if any of the optional items are used, then 3 semi-colons must appear as place-holders (otherwise, a runtime error occurs).

**Example1** +Voice Keyword: en-US;es-MX;;Español [Spanish]
+Voice Keyword: en-US;en-US;;Spanish [Spanish]
+Voice Keyword: es-MX;es-MX;;Español [Spanish]
If Default Language is set to "en-US" then send Espanol to the Spanish recognizer and send Spanish to the en-US recognizer.
But, if Default Lang has been set to "es-MX" then just send Español to the Spanish recognizer.

**Example2** +Voice Keyword: ;es-MX;;Segunda Calle
Always include this and send it to the es-MX speech recognizer

## Form Element (Folder: Elements/Form/)

1. The Form element is the most flexible of the Voice collection elements, allowing you to access and combine multiple types of grammars.

2. VoiceXML has 3 types of grammars:

   A) **Builtin** –The 7 common collection types have grammars that are provided by the gateway and the ASR system: **date, time, digits, number, currency, phone, boolean**.

   Usually these grammars are collected from the caller using the corresponding Studio elements (Date, Time, Digits, Number, Currency, Phone, YesNo_Menu).

   But, for more flexibility, you can specify builtin grammars explicitly in the Form element's Voice Grammar and DTMF Grammar settings.

   This allows you to combine different grammars for voice, DTMF, inline menu, etc.

   B) **Inline** – Inline grammars are created by VXMLServer at runtime and are part of the VXML page sent to the gateway. The gateway passes the grammar to the recognizer. **The recognizer compiles and uses the grammar, but can't cache it for other callers.**

   C) **External** – Custom grammars that reside in a file or a web server are called External Grammars. They are referenced in the VXML page by a URI. The recognizer retrieves the grammars at runtime and caches them for use by other callers.

3. The Form element is also the only Voice element that creates variables indicating the number of noinput and nomatch events that occurred while collecting the caller's information.

# Chapter 5

# The Form Element

- Building inline grammars
- Using external grammars
- Using builtin grammars

4. Ask the caller whether they'd like to
   - **withdraw** funds from (their primary account) **{Data.element.dbAcctInfo.account1}** or
   - **transfer** funds from (their primary account) **{Data.element.dbAcctInfo.account1}** account to (their secondary account) **{Data.element.dbAcctInfo.account2}**.
   - To the caller, this would sound like:

*Would you like to withdraw funds from Savings or transfer funds from Savings to Checking?*

5. Perform the withdrawal or the transfer. For a transfer use the syntax:

```
update account
        set balance1 = balance1 – {Data.Element.getWithdrawAmt.value},
        set balance2 = balance2 + {Data.Element.getWithdrawAmt.value},
        where   acctnum = {Data.Element.getAcct.value}
```

6. Then re-select the both balances from the database and speak them out.

[End of Chapter]

{DATA.SESSION.RETURN0} RXNUM = {DATA.ELEMENT.GETRXNUM.VALUE}; CONCATENATE

MAIN MENU = REFIL

SESSION VARIABLES

# Chapter 8

# Working with Multiple Applications: Invoking Studio Subdialogs and Performing Application Transfers

- The Subdialog Invoke Element to use a Studio Application as a Subroutine
- Writing the Subdialog: Receiving and Returning Data
- Accessing Returned Data
- Application Transfer to transfer control to another Studio Application

# Multiple Studio Applications

For modularization, you may wish to use multiple Studio applications for a single caller.

1. **Subdialogs** - You can write a Studio application as a Subdialog (subroutine) that is called from another Studio application and then returns back. This is analogous to writing a Function that receives parameters and returns one or more results back to the main line code. Except that this function is a Studio application. ICM is unaware of subdialogs being invoked.

    A. Scenario: Studio application Alpha invokes Studio application Beta using a Subdialog Invoke element and may pass data. The VXML Gateway pushes Alpha onto a stack, and uses HTTP to request the URI for the Beta application.

    B. Studio application Beta now executes, sending VXML pages to the Gateway. Beta may do everything available to a Studio application (including invoking another Studio application Gamma as a Subdialog) except it should NOT use a CVP_Subdialog_Return to exit directly to ICM.

    C. When the invoked Studio Subdialog application Beta is done, it uses a Subdialog_Return element to return control back to the application that invoked it. Data may be returned as name=value pairs.

    D. The VXML Gateway now pops Alpha off its stack, and Alpha now continues in the call flow, accessing returned data as Element data of its Subdialog_Invoke element.

2. **Application Transfer** - An Application Transfer is similar to a GOTO statement. It tells VXMLServer to end the current Studio application and start another Studio application for this caller. Data may be passed. ICM is unaware of the Application Transfer.

    A. Scenario: Studio application Alpha collects and validates the caller's account number. Then asks the caller if they'd like to rent a car, book a hotel, or make a flight reservation. Based upon the caller's selection it performs an Application Transfer to another Studio application CarRez, HotelRez, or FlightRez. Alpha may pass along data, such as the caller's account number.

    B. VXML Server ends application Alpha and starts executing transferred-to application and creates Session data for the variables passed from Alpha.

    C. ICM is unaware of the Application Transfer.

    D. Because ICM is unaware of Application Transfers, for reporting and debugging purposes some customers do not use **Application Transfer**, but instead return to ICM to change Call Type and/or send the call back to the appropriate Studio/VXML Server application. In this case you must return data to ICM so it knows where to send the call.

3. **Tracking the call**. Use the folder **VXMLServer/logs/GlobalCallLogger** to track a call through various applications. If the same SessionID appears multiple times, it has visited multiple applications.

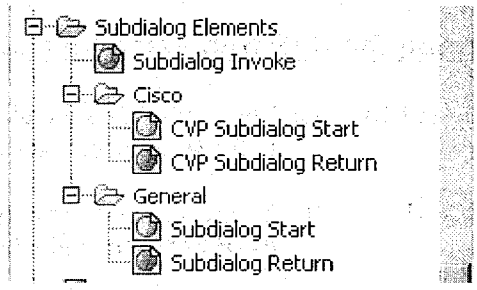
   **Note**: Application names are logged in the order they are EXITED by a caller. So a Subdialog application will appear in the log before the application that invoked it for a caller.

4. **Debugging**: When working with multiple applications, you will have multiple Activity Logs and Error Logs to work with. If an application is invoked or transferred-to and doesn't exist, the error will be logged in the folder **VXMLServer/logs/GlobalErrorLogger**

5. The Call Studio Debugger can not work with multiple applications. You will receive an error if you try to invoke a Subdialog or perform an Application Transfer.

## Studio Subdialogs

(Folder: Subdialog Elements/)




1. Studio Applications can be invoked by other Studio Applications as subroutines to easily create reusable modules which have all the functionality of a Studio Application, such as caller interaction, database connectivity, etc. You can send and return data between the calling application and the subdialog.

2. The **Subdialog Invoke** element initiates a VoiceXML Subdialog call from the VoiceXML Gateway to another or Studio application which **may reside on the same or a remote application server.**

3. While the invoked application is handling the call, the calling application is dormant, waiting for the subdialog to return.

4. Data may be passed to the subdialog. When the subdialog returns, returned data will be stored as Element data of the Subdialog Invoke element.

5. When invoking a Studio application on the same Vxml Server, the same Vxml Server license is used.

6. Limitations:
   A. The Subdialog Invoke element can **not** be tested with the Studio Debugger.
   B. You may not invoke a Subdialog from the main application until you've executed a Voice Element in the application.
   C. The main application must set its **Session Timeout** long enough to allow completion of the invoked Subdialog application.

7. There are two ways to pass data to the Subdialog: As part of the URL query string; or using the **Parameter** setting.

   A. **Query String**: Append **"?name1=value1&name2=value2"** to the query string to pass data as HTTP parameters and automatically store them as Session Data in the Subdialog application, and automatically log them in the Subdialog application's Activity Log.
   **/CVP/Server?application=Payment&_ani={CallData.ANI}&_dnis={CallData.DNIS}**

   B. **The Parameter setting** uses VXML Parameters to pass the data. For each passed parameter, you must configure the Subdialog application to receive and store the data as either Element or Session data.





CREDICARD
Memo VXML Server

NO APPERS ON COMBO BOX

## Configuring the CVP Subdialog Invoke Element

1. **Subdialog URI**

   A) The URI of the subdialog to invoke (relative or absolute URI) - must be accessible to the VoiceXML gateway at runtime.

   B) To invoke another Studio application that resides on the same Vxml Server, enter **/CVP/Server?application=*appname***

   C) (**CVP 7**) You must add the callid (and optionally ani and dnis) as follows for reporting and debugging (and (optionally ani and dnis):
   **/CVP/Server?application=Payment&callid={Data.Session.callid}&ani={Data.Session.ani}}&dnis={ Data.Session.ani }**

   D) (**CVP8**) The callid is passed automatically as it was passed from ICM. But to pass ani or dnis as ICM passes it (so it's stored as Session data and CallData), use the following:
   **/CVP/Server?application=Payment&*_ani={CallData.ANI}&_dnis={CallData.DNIS}***

   PASSANDO VIA VRJ (LOG AUTOMATICO) | CALLID VAI SE INTERVO (ACTIVE LOG)

   **No blank spaces or a carriage return at the end of the query string.**

2. **Local Application**

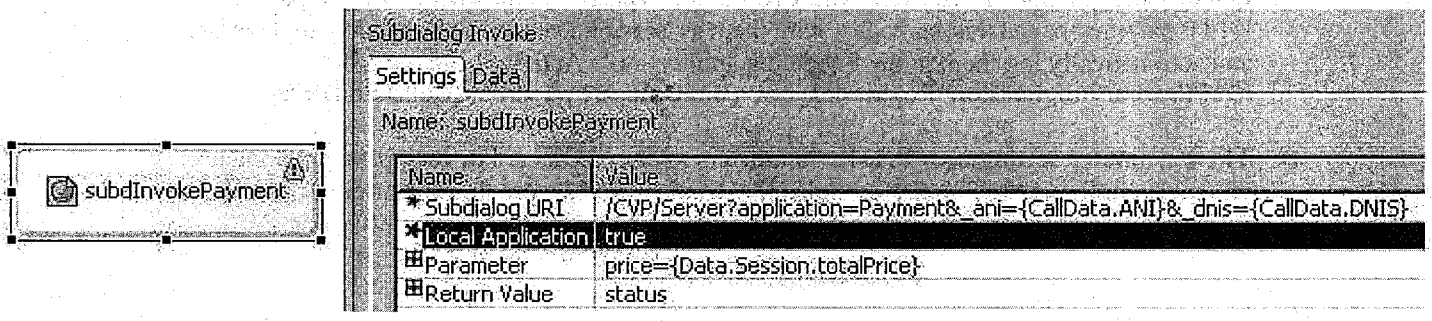
   A) **True** if the subdialog resides on the same Vxml Server. **False** if not.
   B) If invoking a Studio application as a Local Application, then the **same** Vxml Server license will be used. If invoking a Studio application on a Remote Vxml Server uses one license on each Vxml Server.

3. **+Parameter**

   A) Each line holds one ***name=value*** pair of a parameter to pass to the subdialog. Examples:
   **price={Data.Session.copay}**
   B) The name (eg, price) **must** be configured in the subdialog's Subdialog Start element.

4. **+Return Value**

   A) Each line holds the name of one return value from the subdialog. Example: ***status***
   B) The names specified here must match the variable names returned by the subdialog in its Subdialog Return element.
   C) The returned value is stored as **Element data** with this name.
   D) At least one Return Value must exist or a runtime error occurs.










/CVP/SERVER?APP

## Writing the Studio Subdialog

Folder: SubdialogElements/General/SubdialogStart or SubdialogReturn

1. The invoked subdialog application should begin with the **Subdialog Start** element and end with the **Subdialog Return** element.

2. There are no limitations on the functions of a Studio subdialog. In CVP8 you can even invoke another Subdialog from within this Subdialog, and so on.

3. If this application is sometimes invoked by ICM and other times by a Studio application, you can examine the **{CallData.SOURCE}** variable in the subdialog. **{CallData.SOURCE}** either holds the name of the calling application or is blank if the call came from ICM.

4. The subdialog should **NOT** return directly to ICM with CVP Subdialog Return.

### Subdialog Start Element

1. Retrieving data passed in from the calling application:

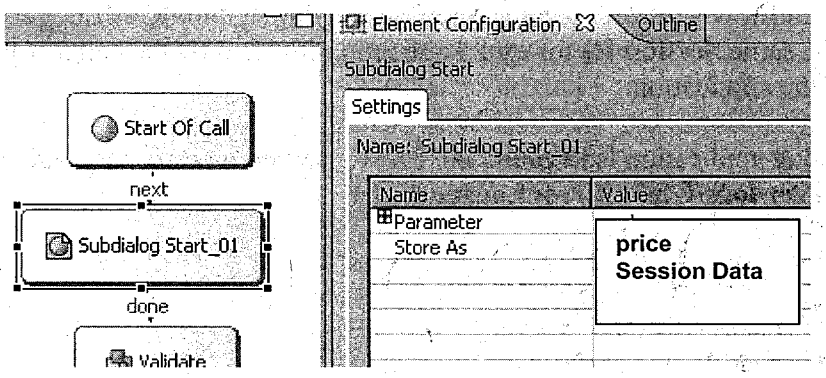
   A. If data is passed from the calling application in a **Parameter** setting, then you **must** configure the **Subdialog Start Parameter** setting with the name. You can store it as Element or Session data. Data is stored as String values.

   B. If data was passed from the calling application **in the query string**, then the data is automatically available as Session data with the same name and logged automatically. Do not include these in the Parameter setting.

**+Parameter**

- The name of one **Parameter** passed as input to the subdialog as a VXML Parameter, ie, specified in the Subdialog Invoke element's Parameter setting.
- It must match the exact value specified in the calling dialog.
- This is a repeatable setting, so multiple values can be specified.

**Store As** Set to store the listed parameters in Session or Element data.



## Subdialog Return Element

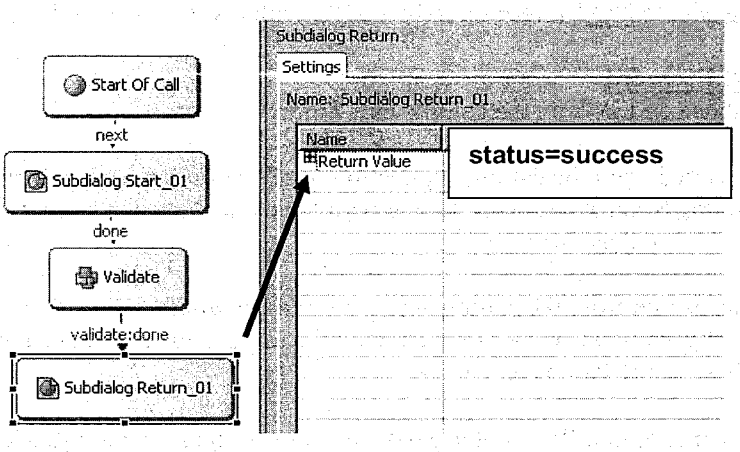
1. To return from the subdialog to the calling application, use the **Subdialog Return** element.

2. To return data to the calling application using the Return Value setting:

**+Return Value** One **name=value** pair per line. Data to be returned to the calling application. Example,
**status=success**

Each **name** must be configured in the calling application's Subdialog Invoke element, in its Return Value setting: Example, ***status*** where it will be stored as Element data of the Subdialog Invoke.

This is a repeatable setting, so multiple values can be specified.




3. **General Subdialog Notes**
   A. The Subdialog Return within a subdialog must return at least one variable, otherwise a runtime error will occur.

   B. CVP8 – You may invoke Subdialogs from within Subdialogs. Each Subdialog ends with a SubdialogReturn – and returns control to the application that invoked it.

   C. To track the callflow you can view the **VxmlServer/logs/GlobalCallLogger** file which has the same **Session ID** for each application invoked by the caller, followed by the name of the application.

   D. If the Subdialog application isn't found on VXML Server, an error message is stored in the **VxmlServer/logs/GlobalErrorLogger** folder

## Application Transfer

1. An **Application Transfer** element allows a Call Studio application to END and a different Studio application (located on the same VXML Server) to start running on this phone line. **This is different than invoking a Subdialog, as there is no return back to the first application.**

2. No telephony transfer is involved, the same VXML Server license is used, ICM is unaware of the application transfer.

3. The sending application's activity log will indicate the application ended as an **application transfer** with the name of the transferred-to application.

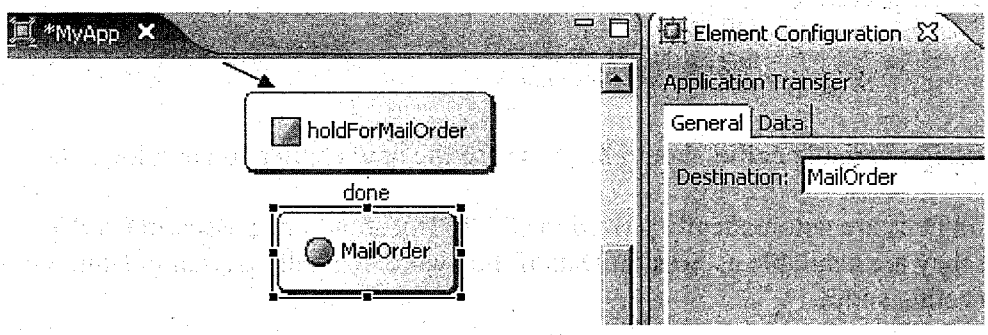
4. Data can be passed from the sending application to the receiving application. The activity log does not indicate whether data was passed (or received into the receiving application).

5. Configure an Application Transfer element by connecting an Application Transfer element to the callflow. In the **Application Transfer** configuration pane, you may either:
   A. Pull down the **Destination** menu and select the name of a Studio application to transfer to. The menu consists of all applications in the Navigator pane.

   B. The Transferred-to application must be a valid application listed in the VXML Server master list of available applications at runtime.

   C. Use the **Substitution** button and specify a **variable** that will contain the name of a destination application at runtime

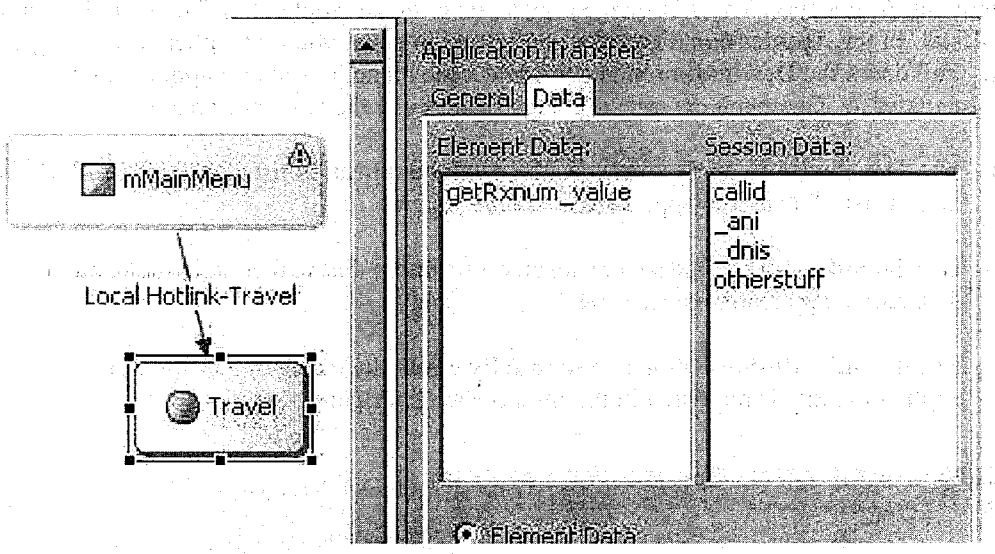
   D. Use the **Substitution** button and **manually enter the name** of a destination application that is not listed in the Studio Navigator pane.



6. You may optionally pass data to the receiving application, using the **Data tab** of the Application Transfer element. You may pass Element data and/or Session data.
   A. For **Element data**:
      a. Select the element name and manually enter the variable name.
      b. Click **ADD**.
      c. The name will appear in the box as **ElementName_VariableName**

   B. For **Session data**:
      a. Manually enter the session variable name.
      b. Click **ADD**
      c. The Session variable name will appear in the box

   C. In the transferred-to application, all passed data will become **Session Data** prefixed by the sending application name: e.g., {Data.Sesion.**MyApp_callid**}




7. The Session variable **callid** should be passed to the new application and logged.

8. Ani and Dnis are automatically passed in CVP8. But should be passed in CVP 7, where they are available as Session Data if they were manually passed in from the ICM routing script.

THIS FORM NOT CREATE ONLY LIST
YOU CAN USE A ONE PREVISYL EXISTS

## The Transferred-to (Receiving) Application

1. When an application starts it can check the variable {**CallData.SOURCE**} to decide if this is a new call or an application transfer.

2. In the Substitution window, select the tab **Call Data** and choose the variable named **Source.**

   - {**CallData.SOURCE**} will either have the **name of the sending application** if this was an application transfer or **will be empty if this is a new call.**

3. Data received by an application transfer is available as **Session Data** in the receiving application, **prefixed with the name of the sending application**, followed by an underscore, followed by the name of the data passed.

   For example, If the application MyApp sends:
   - Element data **getAcct.value**, it becomes Session data {Data.Session.**MyApp_getRxnum_value**}
   - Session data named **callid**, it will become session data named {Data.Session.**MyApp_callid**}

4. While the Source Application name is logged into the Activity Log, the data passed and received by an Application Transfer is **not** automatically logged into the Activity Log, it is the developer's responsibility to log it.

**Activity Log for Travel**

.Travel,01/26/2011 08:59:37.546,,**start,source,MyApp**
. Travel,01/26/2011 08:59:37.546,,start,ani,NA
. Travel,01/26/2011 08:59:37.546,,start,areacode,NA
. Travel,01/26/2011 08:59:37.546,,start,exchange,NA
. Travel,01/26/2011 08:59:37.546,,start,dnis,NA
. Travel,01/26/2011 08:59:37.546,,start,uui,NA
. Travel,01/26/2011 08:59:37.546,,start,iidigits,NA
. Travel,01/26/2008 08:59:37.546,pWelcome,enter,
**//NOTE - this is the result of using the Add To Log function in Studio, otherwise no received data is logged**
**. Travel,01/26/2008 08:59:37.546,** pWelcome,**custom,*** MyApp_getRxnum_value,1111**
. Travel,01/26/2008 08:59:38.874, pWelcome,exit,done
. Travel,01/26/2008 08:59:38.874,Digits_01,enter,
. . .

## Chapter 8 Review Questions

1. What are a few differences between Application Transfer and Subdialog Invoke? That is, when would you use one versus the other?

2. What are 2 ways to pass data to a Subdialog, and for each method, how is the data configured to be received (and where is it stored) in the Subdialog Studio application? Which method logs automatically to the subdialog's Activity Log?

3. How do you pass data in an Application Transfer element? Where is it stored in the transferred-to application? Is it logged automatically?

4. Which global log file shows you a list of the VXMLServer applications that each call has executed, so you can determine if it's executed more than one?

5. In a Subdialog application, which nodes should you begin and end with?

6. With a Subdialog Invoke, are callid, ani, and dnis all passed automatically to the subdialog? If not, which must you pass manually? What is the recommended method of passing this?

7. With an Application Transfer, are callid, ani, and dnis all passed automatically to the new application? If not, which must you pass manually? What is the recommended method of passing this? What is a Best Practice for this once it's received into the transferred to application?

[End of Chapter 8]

# Chapter 9

# Global Commands, Error Handling, RequestIcmLabel, Record, FTP Client, and Multi-Language Applications

- Hotlinks – Enabling Global Key presses And Utterances
- Hotevents – Catching VXML Events
- Error Element – Handling Unrecoverable Errors
- Request ICM Label – Interfacing to ICM Within the Studio Call Flow
- Record (with Confirm) – Taking a Recording From the Caller
- FTP Client Element
- Multi-Language Applications - Using Application Modifier Element

## Hotlinks and Global Commands

1. A **Global Hotlink** is a globally accessible utterance and/or key press that immediately brings the call to a specific part of the call flow or immediately throws an event as configured. Any number of Hotlinks can be used in an application and are active throughout the entire application.

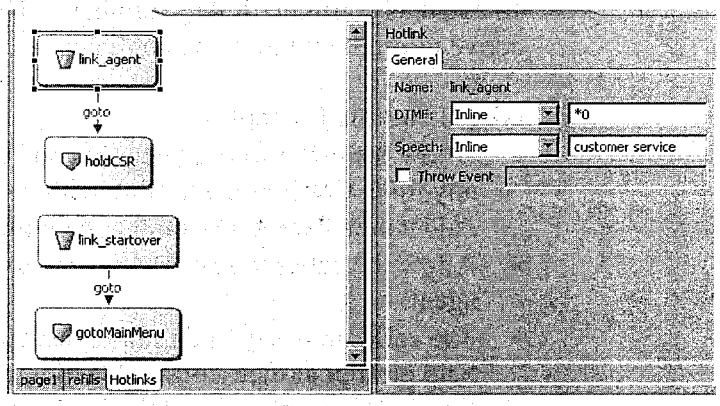
2. Global Hotlink Examples:
   - Allow the caller to say "main menu" or press "*1" to return to the main menu at any time. Configure a hotlink with an exit state that connects to a Page Connector. Add a new Page Entry on the main page that points to the MainMenu.
   - Allow the caller to say "customer service rep" or press "*0" to transfer to an agent.
   - The caller may say "I do not understand" or press "*2" to hear a context sensitive help prompt at their current location in the call flow. Configure a hotlink to throw the "**help**" event and add the **HELP** audio group to each Voice Collection element.

3. **Local Hotlinks** can be enabled temporarily within each Voice collection element to allow the system to collect something other than what's being collected in that element.

4. **Local Hotlink Examples:**
   - Add another 'menu' option
   - Allow a caller to enter multiple touchtones at a menu
   - Allow callers to enter a star key when collecting digits

## Global Hotlinks

1. Global Hotlinks are not part of a call flow, they may reside on a separate page without an Entry Connector. The Hotlink is created as VXML code in the **application's root document**.




2. To configure a Global Hotlink:

   A. Specify whether you are using an **Inline** DTMF key sequence or an **External** DTMF grammar to specify the key sequence to trigger this Hotlink. Then enter the DTMF sequence or the URI.

   B. Specify whether you are using an **Inline** speech sequence or an **External** speech grammar to trigger this Hotlink. Then enter the string of words or the URI of the grammar. Synonyms should be comma-separated.

   C. A Global Hotlink can either redirect the callflow by connecting its exit state '(this is the default) or it can be configured to throw an event.

   D. Alternatively, the Global Hotlink can throw an event by selecting the **Throw Event** checkbox and entering an event name. Events can be standard VXML Events (help) or can be custom events that you catch using a Hotevent element.

## Disabling Hotlinks (Global and Local)

Each Voice collection element has a Setting named **Disable Hotlinks**, that allows you to **disable all Global and Local Hotlinks** temporarily while executing that element.

This is useful if the hotlink word or keypress conflicts with the one being collected within that voice element.

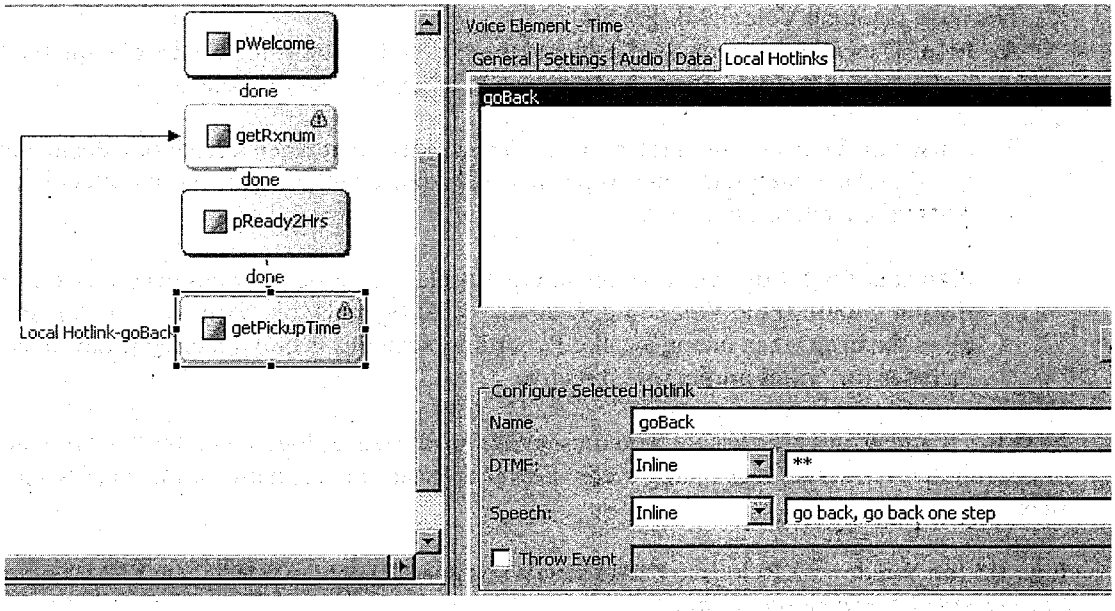
## Local Hotlinks

1. Studio allows you to create **Local Hotlinks** on the Voice collection elements.

2. This provides local enabling and handling of caller commands.

3. For example, allow the caller to say '***go back***' and connect the ***hotlink:goBack*** exit state to the previous collection element.

4. Each hotlink can either throw an event (which must be caught using a Hotevent element or by Vxml Server) or create a new exit state.




5. In a Voice collection element, select the Local Hotlinks tab,
   A. Press the **Add** button, this creates a new hotlink entry in the window
   B. Select the hotlink **New_Local_HotlinkN.**
   C. Configure it as you would a global hotlink
      i. Name: rename the hotlink (eg, **goBack**)
      ii. Specify DTMF sequence to trigger this hotlink
      iii. Specify a comma-separated set of utterances the caller can use to enable the hotlink if you have ASR
   D. If you select **Throw Event**, then enter a Vxml (or user-defined) event name
      i. If you do not Throw an Event, a new exit state will be created named **Local Hotlink - *<hotlinkName>***
      ii. If you do throw an event, create a HotEvent element to catch the event.

## Hotevent Elements to Catch VoiceXML Events

1. A **Hotevent** is an event handler that defines code to execute if the specified VoiceXML event is thrown by the VoiceXML Gateway.

2. Each Hotevent specifies one event. Any number of Hotevents can be used in an application.

3. At runtime, the Hotevent appears in the VoiceXML Root Document as a <catch> element.

4. To Configure a Hotevent, specify a VXML event name (case-sensitive) and configure an exit state. For a full list of Cisco VXML errors and events, see the ***Cisco VXML Programming Manual.***

5. The most common VoiceXML Events:
   A. **help** The caller has requested help and the system handles it for you by playing the Help audio groups that you've configured in each of your Voice Elements.

   B. **error.badfetch** The interpreter context throws this event when a fetch of a document has failed *and* the interpreter context has reached a place in the document interpretation where the fetch result is required.

   C. **error.badfetch.http** (error.badfetch.https) An http specific error has occurred (e.g. audio file not found). If the VoiceXML Gateway is configured with 'vxml version 2.0' so it plays the beep when taking a caller Recording, then this is only reported if the Gateway includes the **'vxml audierror'** directive.

**NOTE** – If audio files are missing, you must either enable debug logging on the VXML Gateway or consult the web server logs on the Media Server to determine the name of the missing audio files.

E.g., IIS (on Windows XP) logs to **C:\WINDOWS\system32\Logfiles\W3SVC1\exYYMMDD.log**

**Standard HTTP Response codes**

- **4xx – error.** (**404** file not found)
- **2xx – successful** (**200** download successful)
- **3xx** – further action may be needed (**304** file not modified since last Gateway download)
- **5xx** Web Server errors

**Gateway Debug Command**

- term mon
- **debug http client error**
- **<call into the system to view the missing files>**
- u all (disable all logging)

D. **error.semantic** A run-time error was found in the VoiceXML document, e.g. an invalid character, an invalid VXML Property value, or an invalid DTMF character when using the DTMF Gateway.

E. **error.noresource** A run-time error occurred because a requested ASR, TTS, or Media Server resource was not able to function properly. This could be due to a missing external speech grammar, or a grammar that contains invalid entries, occasionally it indicates missing audio files.

As of 12.4(15)T the **error.noresource** event replaces these events (which are supported for backward compatibility):

- **error.com.cisco.resource.failure.asr** (the ASR failed to find or load the specified grammar),
- **error.com.cisco.resource.failure.tts** (the TTS failed to speak the requested text)
- **error.badfetch.http** (the VoiceXML Gateway failed to retrieve or play prerecorded audio files).
- **error.com.cisco.media.resource.unavailable**

F. **error.unsupported.format** The requested resource has a format that is not supported by the platform, e.g. an unsupported grammar format, or media type.

G. **error.unsupported.language** The platform does not support the language for either TTS or ASR.

BADFETCH → TIMEOUT OR MEDIA SERVER DOWN OR CAN'T CONNECT

HTTP://BULLSHIT/***.WAV MSG COM SERVER NÃO EXISTENTE

BADFETCH.ERR.0.HTTP → AUDIO FILE OR SOMETHING ELSE DO NOT EXIST OR

ERRO ACHAR BANCO JNDI PARA INICIALIZAR

SAY SMART DATA PRICE

TYPE CREDIT CARD



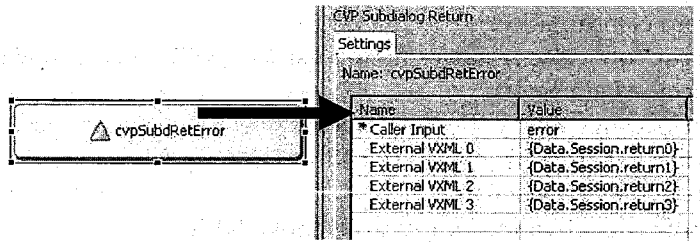


É POSSIVEL ALTERA EM CASO DE O MEDIA SERVER ESS MUDANÇA E FEITA NO ICM

## Error Element

1. When an unexpected (**unrecoverable**) error occurs, the application can be configured to send the call and collected data back to ICM in CVP Comprehensive model; or to transfer the caller to an agent in CVP Standalone models. This is accomplished by creating an **Error element**.

2. If you have no Error element in the application, an unrecoverable error causes the call exit back to ICM, and continue down its error path and no data can be returned to it.




3. Unrecoverable errors include both Java exceptions (e.g., Say it Smart exceptions) and VoiceXML events (such as error.badfetch) that you are not handling with a HotEvent element.

4. Since the error is **unrecoverable**, the VXML Server portion of the call ends after the Error element completes.

5. The Error element causes the following to occur:
   - Suppresses playback of the error message configured under Project/Properties/Audio
   - Executes one Studio element designated as the Error Element. This would be a **CVP Subdialog Return** in CVP Comprehensive model to return data to ICM; or a **Transfer** element in CVP Standalone to transfer the caller.
   - Causes the call to be marked as a 'success' in the Activity Log and returns down ICM's success path.

6. An application can have only one Error element.
   A. Drag a CVP Subdialog Return, Transfer, or Audio element into workspace and configure it normally.
   B. **Right-click** on it, select the **Error Element** option, and select **Yes**.
   C. The shape on the element will change to a triangle.
   D. An Error element has no exit states and is not connected to any elements in the callflow.
   E. Use the word 'Error' in the name of the element, as the call will **not** be marked as ending in an error in the Activity Log or ICM.
   F. If returning data to ICM, use Caller Input to indicate that an error occurred because ICM will have to explain that there's a problem to the caller before queuing. You should also have variables containing data in a format to return to ICM already configured by concatenating or assigning data throughout the callflow as the data is collected.

G. To have data available to return to ICM in an Error Element, create Session variables return0, return1, return2, return3 in a Data tab at the beginning of the application.

Decide upon the format of data being returned to ICM, e.g., perhaps you'll be using = and ; as delimiters:

;menu=refills;rxnumber=1111;pickuploc=boston;

H. Use the Data tab of an ApplicationModifier (**dummySetVar**) to create Session variables return0, return1, return2, return3 in a Data tab at the beginning of the application. Assign each the initial value of your delimiter (;)

I. As you collect information that is important to return to ICM upon an error occurring, use an ApplicationModifier element's Data tab to continually concatenate the new information onto the end of the existing {Data.Session.return0} variable. Name the AppModifier element **dummySetVar1 (or 2)**

After the mMainMenu:
Session Data
Name: return0
Value: {Data.Session.return0}menu=refills;
Press: Add
This would return something like this to ICM **;menu=refills;**

Copy/Paste the dummySetVar1 and paste it after the caller confirms their Rxnum. In the Data tab, highlight the existing **return0** in the upper right box:
Session Data
Name: return0
Value: {Data.Session.return0}rxnumber={Data.Element.getRxnum.value};
Press: **Update**
This would return something like this to ICM **;menu=refills;rxnumber=1111;**

## Record With Confirm Element (Folder: Elements/Record/)

1. The **Record** element allows you to take a recording or to update audio prompts on the phone.

2. The gateway plays a prompt to the caller and takes a temporary recording. Once the recording is completed (or confirmed) it is sent to the VXML Server for storage and deleted from the gateway. Note that recordings aren't deleted from the gateway until the call ends.

3. Recording requires that the Ingress Gateway dial peer contains the directive '**codec g711ulaw**' instead of voice-class 1, which allows other codecs, otherwise '**error.noinput**' occurs.

4. The **Record_With_Confirm** element allows the caller to listen to, and either accept or re-record the recording.

Settings

1. **Input Mode** The type of entry for the confirmation.

2. **Noinput Timeout** The maximum silence before a noinput event is thrown. This is only enabled during the recording if the Ingress Gateway dial-peer does not contain '**no vad**'

3. **Record Max NoInput Count** The maximum number of noinput events allowed during recording capture. 0 = infinite noinputs allowed. This requires the Ingress Gateway directive '**no vad**' is absent from the incoming dial peer.

4. **Max Disconfirmed Count** The maximum number of times a caller is allowed to reject a recording and to re-record. **NOTE** – If allowing a large number of Disconfirmations, the Session (Inactivity) Timeout may need to be very large and enough space must be allocated on the gateway to store all the recordings until the call ends.
   **ivr record memory session <KB>** (max size per call, 500KB approx'y 1 min.)
   **ivr record memory system <KB>** (concurrent recording sessions)

5. **Start With Beep** Whether or not the VoiceXML Gateway should play a beep before recording begins.

6. **Terminate On DTMF** Whether or not the caller can end the recording by pressing a touchtone key.

7. **Keep Recording On Hangup** Whether or not the recording is stored by VXML Server if the caller hung up during the recording or the confirmation menu. Default = false.

8. **Max Record Time** The maximum time allowed for the recording. Include time unit (s for seconds, ms for milliseconds).

9. **Final Silence** The interval of silence that indicates the end of speech (alternatively, the caller presses a DTMF tone to end the recording if Terminate on DTMF is set to true)

10. **Filename** The filename of the recording (without extension). If left blank, an auto-generated name **audio*N*.wav** where N is the number of milliseconds since midnight January 1, 1970 (GMT) is assigned.

11. **File Type** This specifies the audio type of the file that will hold the recording.

12. **Mime Type** (Optional) This specifies the MIME type of the file that will hold the recording, if file type is set to other.

13. **File Extension** (Optional) This specifies the file extension to use for the recorded file. Omit this if the extension is desired (eg, .wav for wav files)

14. **Path** The path to the file that will hold the recording. **If left blank, the file is assumed to be sent via ftp.**

15. **FTP Host** The domain name of the host to ftp the recording. **If left blank, the recording is assumed to be stored in a local file.**
16. **FTP User, Password** The user and password to use while FTPing the recording.
17. **FTP Path** The directory into which to FTP the recording.
18. **FTP In Background** Whether or not the FTP is to be performed in the background.

## Audio Groups

**Record Confirmation Audio Groups:**

1. **Before Confirm** Played before the recording is played back. The recording will be played back after this audio group is done playing.
2. **After Confirm** Played after the recording is played back. At least one of the two confirm prompts must be specified.

## Element Data

1. **filename** This stores the filename of the recording (without the path). For example **mymessage.wav**

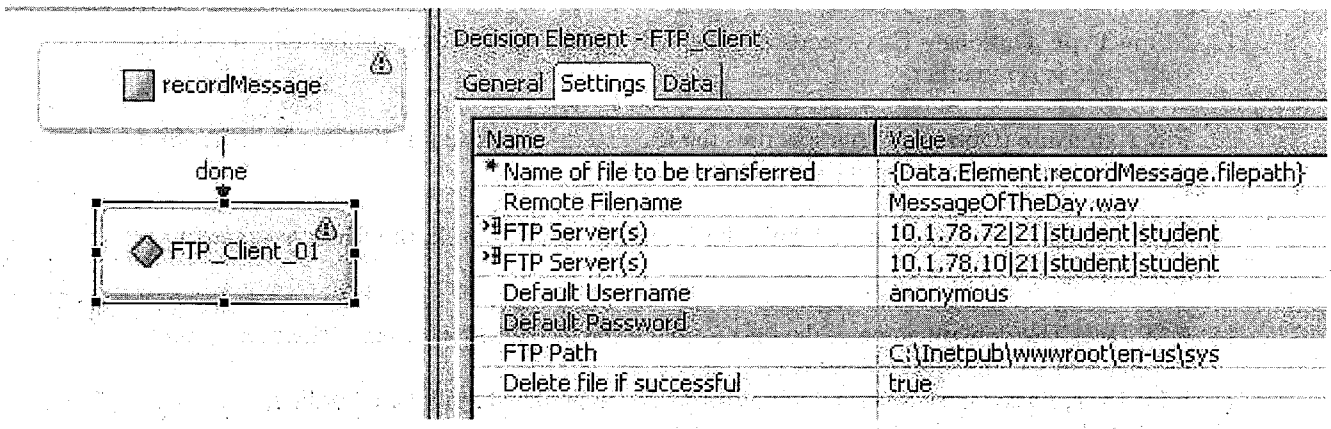
2. **filepath** This stores the path to the file holding the recording (including the filename). For example, **C:\Inetpub\wwwroot\en-us\sys\mymessage.wav**

3. **confirm_confidence** This is the confidence value of the utterance for the confirmation menu.

4. **maxTime** (boolean) – This stores a "true" if the recording ended because the caller exceeded the maximum duration.

## FTP_Client Element (CVP 8.5) (Folder: Elements/Integration/)

1. The **FTP_Client** element is used to upload a local file from VXML Server to one or more FTP servers concurrently. For example, use the **FTP_Client** element after the Record element to ftp audio recordings to multiple Media Servers.




2. **Files are FTP'ed in the foreground**. Consider setting the VXML Property **'fetchaudio'** so that the caller hears music while the FTP operation is in progress.

3. If an FTP file transfer takes longer than the gateway's configured **http client response timeout** in seconds, the FTP transfer will complete correctly, but the call will drop as soon as the configured timeout duration is met.

   To prevent this, set the VoiceXML Property **fetchtimeout** to a large time unit (e.g., **120s**) in the Voice element right before the FTP_Client element.

4. Ensure the FTP Servers allow **write** access.

| Settings | |
|---|---|
| **Name of File** | Full Windows directory path and name of the file in standard Windows format. |
| **Remote Filename** | (Optional) Name of file on FTP Server. If omitted, current file name is used. |
| **FTP Server(s)** | This is the list of FTP server host names or IP addresses to transfer the file to.<br><br>Each FTP server entry may optionally specify a port number (default port:21), username and password in the format **host\|port\|username\|password. If username and/or password are not included, the Defaults (below) will be used.**<br><br>Server entries are delimited by a space character, entering multiple hosts on one line or you may enter them on separate lines or both.<br><br>Escape codes must be used if any field requires spaces (use **\s**), vertical bars (\|) (use **\p**) or equals symbols (=) (use **\e**). |

| | |
|---|---|
| **Default Username** | Username to use. This may be overridden on a per server basis by entries in the FTP Server setting. |
| **Default Password** | Password to use. This may be overridden on a per server basis by entries in the FTP Server setting. If Password is specified, Username must also be specified. |
| **FTP Path** | This is the directory on the FTP server where to transfer the file.<br><br>Use the Windows format of a forward slash as the subdirectory delimiter **dir/subdir**. The directory will be created if it does not already exist. |
| **Delete file if successful** | Whether to delete the file after successfully transferring to **all** FTP Server(s). |
| **Element Data** | |
| | Element Data is **only created** when an error occurs with one of the FTP operations. That is, when the exit state is **not done** |
| **failed_servers** | One or more space delimited host names or IP addresses of Server(s) where the input file was **not** successfully transferred. This data is created only if the exit state is not done. |
| **failed_servers_ reasons** | One or more space delimited reason codes indicating why a file was not successfully transferred. This data is created only if the exit state is not done.<br><br>•**connection_error**: There was an error connecting to the FTP server. This may be caused by an invalid or blocked port.<br>•**extraneous_data**: There were extra fields for a given server in the ftp_hosts setting.<br>•**invalid_filename**: The name of the file to transfer is invalid or the file doesn't exist.<br>•**invalid_port**: The port for an FTP server is invalid.<br>•**missing_username**: The password for an FTP server was specified, but the username was left blank. They must either both be specified or both left blank.<br>•**unknown**: An unknown error has occurred.<br>•**unknown_host**: An FTP server could not be reached. Possible reasons include an incorrect hostname or network connectivity problems.<br>•**three-digit number**: An FTP server sent back an unexpected reply code. Additional information will appear in the error log.<br>•**Java exception**: An unexpected exception was handled. See error log for details. |
| **Failed_servers_ count** | Number of failed FTPs. This data is created only if the exit state is not done. |
| **Exit States** | |
| **Done** | This exit state is used when all FTP transfers were successful |
| **Partial_success** | This exit state is used when at least one, but not all FTP transfers were successful. See Element data for number of failed FTPs, list of failed servers, and reasons. |
| **Error** | The file was not transferred to **ANY** FTP Server. See Element data for list of failed servers and reasons. |

## IIS Expiration Configuration

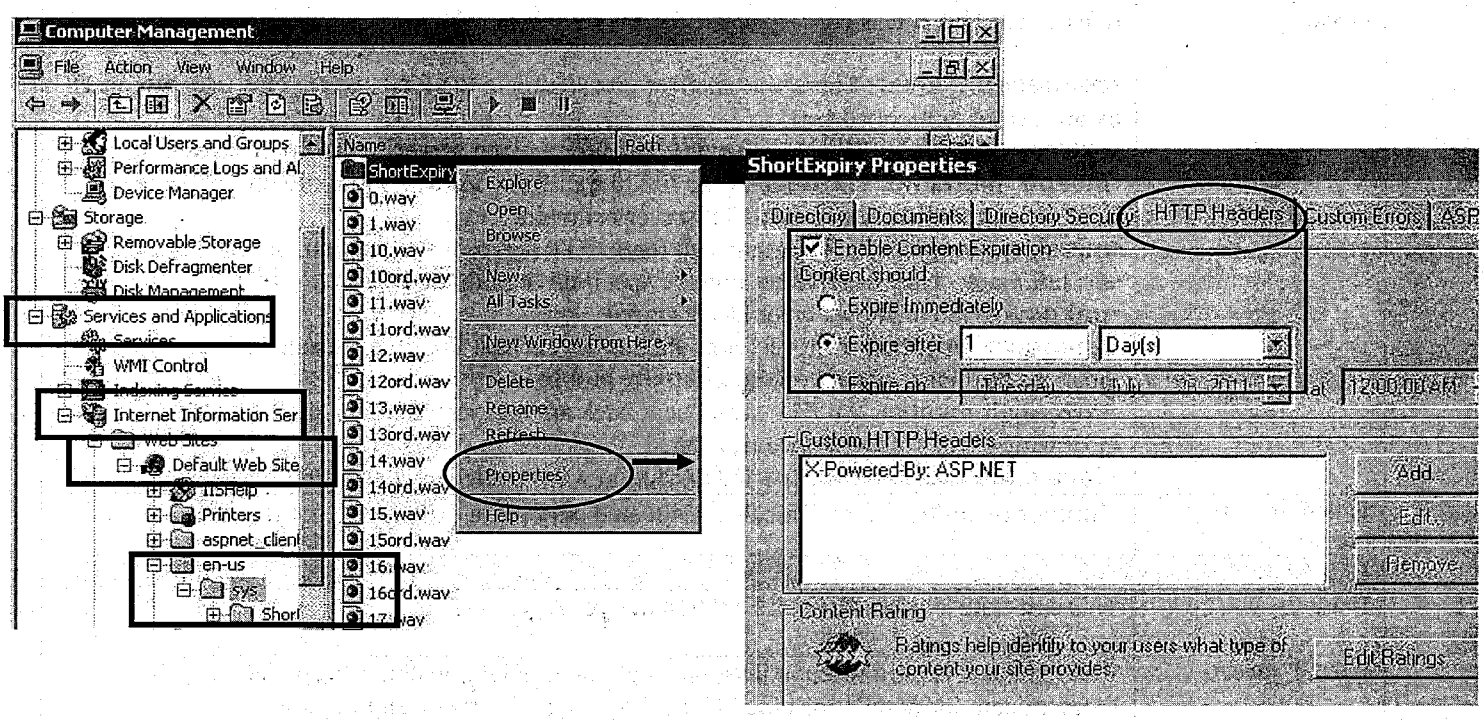
1. If you are using the Record element to modify outgoing audio prompts, then you should configure the media server so the audio file expires from the gateway's cache within a reasonable amount of.

2. This configuration should be done ahead of time on the media server. Otherwise, you'll have to perform the **clear http client cache, set http client cache stale,** or **audio load <URI>** from the gateway to force the new audio file to be loaded.

3. Recall from Chapter 2, IIS allows you to configure **cache expiration** for each directory or file.

4. Note that if the media server file hasn't changed, it will not be downloaded even after it's expired.

5. To configure IIS on the media server, launch the **Computer Management** tool:
   Press **Start** (Bottom-Left of your PC) >
   Select **MyComputer** (Right-click)>
   Select **Manage**



## Application Modifier Element

1. The **Application_Modifier** action element is used to modify environment settings and remove session data values at runtime in an application.

2. A typical use for the Application_Modifier element would be for multi-language support because it can be used to change the **ASR and TTS Language** and **Default Audio Path**. Note that these settings may be specified initially in the Project / Properties windows.

3. The Application_Modifier element updates the application for the current session only.

4. Settings:

- **Maintainer** - This optional setting specifies the e-mail address of the voice application administrator or maintainer. This value is set in a VoiceXML <meta> tag.

- **Language** – This setting specifies the ISO language-locality identifier to specify in each VoiceXML document's "xml:lang" attribute. It is used in determining the ASR and TTS engines to use.

  Some common language identifiers are en-US (US English), en-GB (Great Britain English), es-MX (Mexican Spanish), fr-CA (French Canadian).

  This value is set in the <vxml> tag of each generated VXML page for example:

  **<vxml version="2.0" xml:lang="en-US" >**

- **Encoding** - This setting specifies the encoding (character set encoding) to use when creating VoiceXML documents. This value is set in the <xml> tag, for example the default is:

  **<?xml version="1.0" encoding="UTF-8"?>**

  a) XML documents can contain non ASCII characters, like Norwegian æ ø å , or French ê è é. To avoid errors, specify the XML encoding, or save XML files as Unicode.
  b) Some standard encoding formats are: ISO-8859-1, UTF-8 (8-bit encoding, the default), UTF-16 (double-byte encoding).

- **Default Audio Path** - This setting specifies a partial or full URI to a path containing the audio content for this voice application. This is often changed to reflect different language prompts.

  For example,

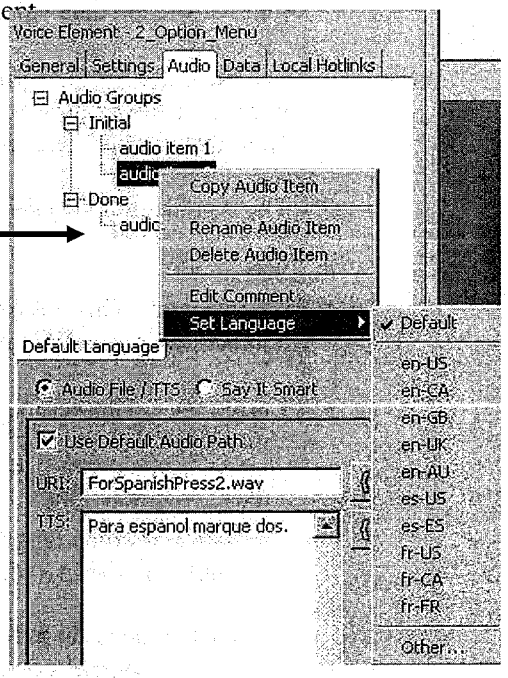
  **http://10.1.78.12/audio/en-us/sys/ (for English)**
  **http://10.1.78.12/audio/es-mx/sys/ (for Spanish)**

- **Session Data to Remove** - This setting specifies the names of session data values to delete from this voice application.

## Mixed Language Prompts

1. Beginning with CVP 4.1/7.0, you can add language tags to individual audio TTS items or distinct audio prompts for different languages within one Voice element.

2. To create a language menu that speaks (for example) both English and Spanish in one element, do the following: **right-click the audio item**, select **Set Language >** specify the ISO language setting (2 lower-case letter locale – 2 upper-case letter language) for the TTS.



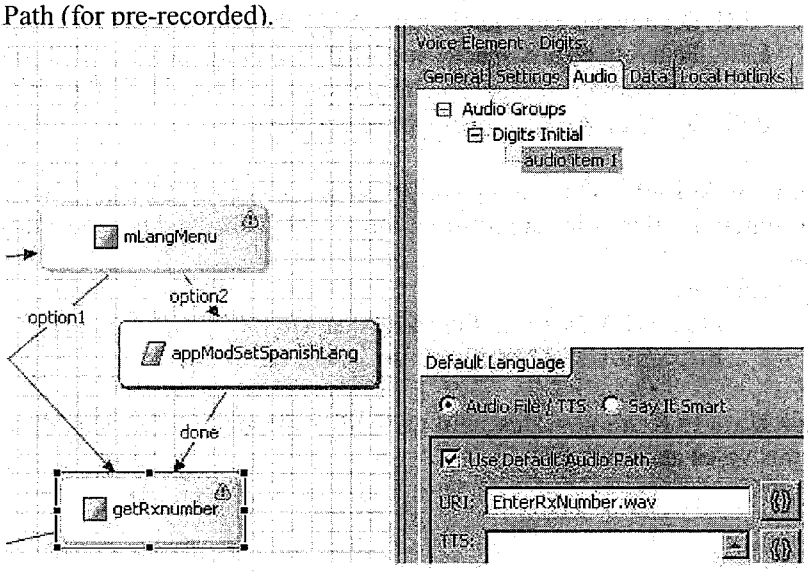

3. For example, to present the caller with a bilingual menu:

**audio item1:**
URI: (Use Default Audio Path) **ForEnglishPress1.wav**
TTS: **For English press 1**

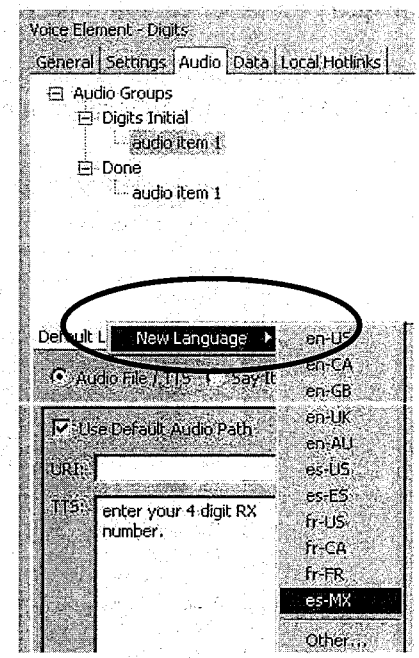
**audio item2: (Right-click, Set Language> es-MX)**
URI: (Use Default Audio Path): **ForSpanishPress1.wav**
Recorded in Spanish, but in the en-us directory.
TTS: **Para espanol marque dos**

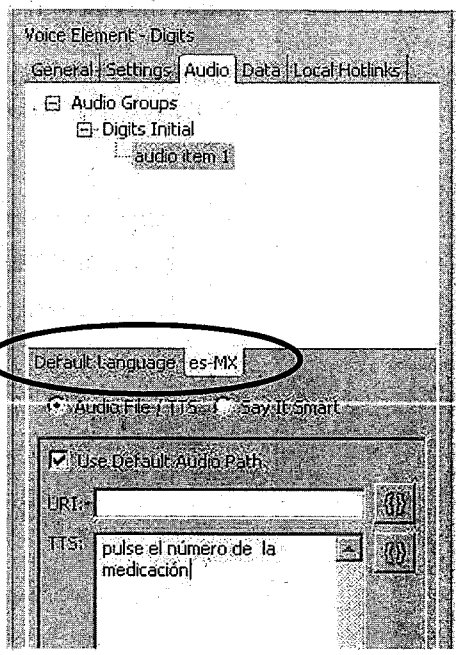
4. If the caller selects Spanish, use the **Application Modifier** element to change the following:
   - Default Audio Path **http://IPaddress/es-mx/sys/**
   - Language **es-MX (for ASR/TTS/AudioPrompts)**

5. In the succeeding voice elements, you can use the Default Language tab and the modified Default Audio Path (for pre-recorded).



6. Since TTS Text must be in the appropriate language, you can specify entirely different prompts for different languages, by **right-clicking** on the **Default Language** tab and select **New Language**



## **ReqICMLabel** (Folder: Elements/Cisco/)

1. A VXML Server Application can now interact with an ICM Script during its callflow rather than just at the end of the application to query for expected wait time, etc.

2. The **ReqICMLabel** element allows the Studio application to pass data into ICM ECC Variables: Peripheral Variables 1-10, FromExtVXML[0],…,[3], and caller_input.


Decision Element - ReqICMLabel

General | Settings | Data

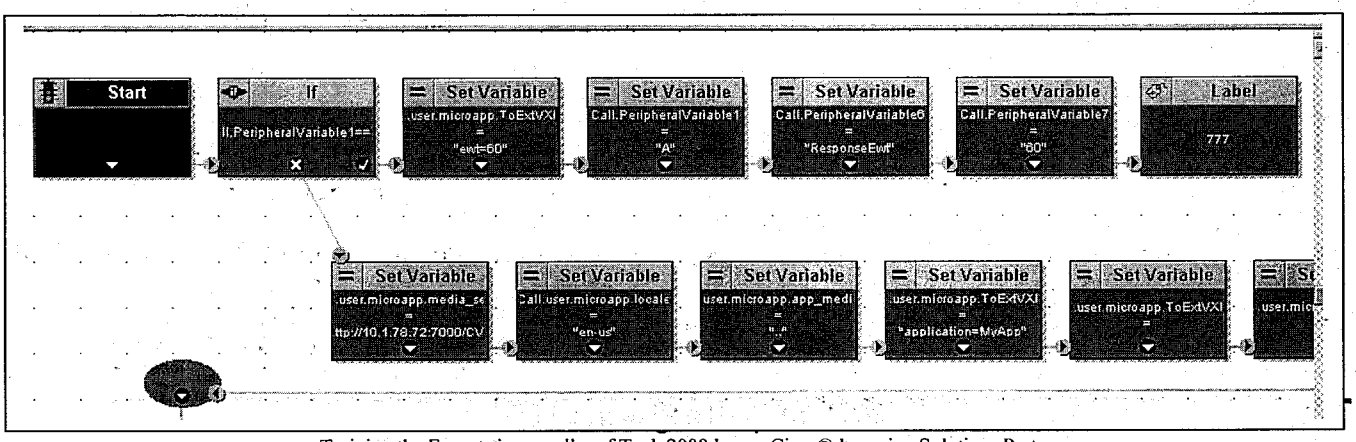
| Name | Value |
|---|---|
| * Timeout | 3000 |
| callvar1 | 1 |
| callvar2 | skill |
| callvar3 | sales |
| callvar4 | language |
| callvar5 | spanish |
| callvar6 | request |
| callvar7 | ewt |
| callvar8 | |
| callvar9 | |
| callvar10 | |
| FromExtVXML0 | a |
| FromExtVXML1 | b |
| FromExtVXML2 | c |
| FromExtVXML3 | |
| caller_input | test |

3. This request is passed directly from **VXMLServer to the CallServer to ICM** (bypassing the Vxml Gateway).

4. ICM starts running a **new instance** of the ICM routing script **for the original DNIS.**

5. Since this same script runs upon a new call and upon a ReqICMLabel, you must indicate to ICM that this is not a new call but a ReqICMLabel. For example, pass a value such a '1' from VxmlServer into the PeripheralVariable1.

6. You should consider how this might impact ICM reporting.



7. The data passed from VXMLServer is accessible in ICM ECC variables (PeripheralVariable 1-10, caller_input, and FromExtVXML[0],…[3]).

8. ICM can then assign values to return to VXML Server in:
   - **Peripheral Variables 1-10** - These will be stored into Element Data of the ReqICMLabel element, and named **callvarReturn1**, . . , **callvarReturn10.**
   - **ToExtVXML[0],…,[4]** in the format ***name_i=value_i*** (multiple *name=value* pairs in one array entry must be semi-colon separated) – these will become **Session variables** in the Studio application with these names and values.
   - The ICM script ends with a **Label node** – the returned label will be stored into Element Data of the ReqICMLabel element, and named '**result**'.

9. When the ICM script must end with a **Label node.**

10. The VXML Server application continues down the ReqICMLabel node's exit state **done** (or error). The following is a VxmlServer Activity Log displaying the result of a ReqICMLabel node.

```
9,Audio_01,exit,done
9,ReqICMLabel_01,enter,
9,ReqICMLabel_01,custom,doDecision,777
9,ReqICMLabel_01,custom,***,expectedWaitTime=60
9,ReqICMLabel_01,data,callvar1,1
9,ReqICMLabel_01,data,callvar2,skill
9,ReqICMLabel_01,data,callvar3,sales
9,ReqICMLabel_01,data,callvar4,language
9,ReqICMLabel_01,data,callvar5,spanish
9,ReqICMLabel_01,data,callvar6,request
9,ReqICMLabel_01,data,callvar7,ewt
9,ReqICMLabel_01,data,callvar8,
9,ReqICMLabel_01,data,callvar9,
9,ReqICMLabel_01,data,callvar10,
9,ReqICMLabel_01,data,FromExtVXML0,a
9,ReqICMLabel_01,data,FromExtVXML1,b
9,ReqICMLabel_01,data,FromExtVXML2,c
9,ReqICMLabel_01,data,FromExtVXML3,
9,ReqICMLabel_01,data,caller_input,test
9,ReqICMLabel_01,data,result,777
9,ReqICMLabel_01,data,callvarReturn1,A
9,ReqICMLabel_01,data,callvarReturn2,skill
9,ReqICMLabel_01,data,callvarReturn3,sales
9,ReqICMLabel_01,data,callvarReturn4,language
9,ReqICMLabel_01,data,callvarReturn5,response
9,ReqICMLabel_01,data,callvarReturn6,ewt=60
9,ReqICMLabel_01,data,callvarReturn7
```

**Element Data Created**

**callvar1-10** – data passed to ICM Peripheral Variable 1-10

**FromExtVxml0-3** passed to ICM
**caller_input** passed to ICM

**result** - Label returned from ICM

**callvarReturn1 – 10** values of Peripheral Variables 1-10 after returning from ICM are stored here.
**callvar1-10** (above) - Peripheral Variables sent to ICM (with the original values)
**SessionData (not logged by default)**
ToExtVxml[i] ***name=value*** pairs – each pair becomes a Session variable with that **name** and **value**

## Chapter Review Questions

1. Which VoiceXML event would each of the following cause:

   A. Gateway tries to retrieve an audio URI that is missing

   B. The ASR grammar represented using a URI can't be found and the ASR complains about it

   C. The VXML Server is too slow returning the next page to the gateway because some back end database processing is taking 60 seconds

   D. A VoiceXML syntax error has occurred due to an incorrect setting in a Menu element

2. How do you create an Error element? When does it execute?

3. What is the benefit of having an Error element?

4. Suppose you have a global HotLink in MyApp to allow the caller to return to the Main Menu at any time. Explain what you would have to do to allow this functionality while the caller is in Subdialog that's been invoked from MyApp. That is, how would you
   a) Listen for MainMenu in the Subdialog application;

   b) Take the caller back to the Main Menu in the MyApp application

[End of Chapter 9].

# Chapter 10

# Call Studio: Working with Custom Java Components

- Introduction to the CVP Java and XML APIs
- Deploying Custom Java Components
- Creating a Java Project in Eclipse
- Implementing Custom Components
- Standard Actions
- Standard Decisions
- Configurable Action and Decision Elements
- On Start of Call and On End of Call
- Dynamic Configurations for Dynamic Menus
- Say it Smart Plugins

## Introduction to the Custom Java Components

Programmers may extend the functionality of the Call Studio by building **custom components** using the **Cisco CVP Java API**. The most common custom components developers create are discussed below:

1. **Standard Actions** and **Standard Decisions**
   - Drag in an element named 'Action' (or 'Decision') and enter the name of a Java class or URI to execute at that point in the callflow.
   - Best for specific 1-time usage as no 'parameters' can be passed to them although they have access to data by name

2. **Configurable Action** and **Configurable Decision** elements (Java only)
   - These create new elements in the Element pane.
   - Self-documenting and have a Settings tab for maximizing reusability
   - Settings can be used to pass variables

3. **Dynamic Configurations** uses Java to modify or create the Settings and Audio at runtime based on caller-specific information (eg, Dynamic Menus)

4. **Start of Call** - Execute at the start of each caller's visit to the. Often used to create Session data or set default audio path, language and root document properties

5. **End of Call** - Executes at the end of the caller's visit to the application (regardless of how it ends). Often used to create Call Detail Records

6. **Say It Smart Plugins** - Say it Smart Plugins render information into an array of audio files (and TTS content) to play. Often used for Ordinals, Alphanumerics, Foreign Language currency say-it-smarts.

7. **Start of Application (End of Application)** – Executes when VXML Server loads/unloads the application from memory, e.g. upon reboot or updateApp script. Usually used to create (or cleanup) 'Global' data available across all calls

8. **Voice Elements** - These create new Voice elements in the Elements pane to interact with the caller. These are difficult to write and require some knowledge of VoiceXML.

## Deploying Custom Java Components

Custom Components can be restricted to one application (**Application Specific**) or can be made available to all applications running on VXML Server (**Global**).

### Application Specific Deployment of Custom Java Components

**Call Studio (Application Specific):**

1. Only the compiled Java code is required. A single Java **class** file has the **.class** file extension. A library of multiple class files has the **.jar** file extension.

2. In the Studio Navigator pane, copy the compiled Java code into either:
   i. **<appname>/deploy/java/application/classes/ or**
   ii. **<appname>/deploy/java/application/lib/**

   Note that **Studio only requires the Java classes for custom Configurable Elements and Say it Smart plugins**. All other custom components need only be deployed on VXML Server, although you might deploy it from Studio to VXML Server for convenience.

3. When you deploy the application from Studio, the Java classes will deploy to VXML Server into the appropriate directory (listed below).

4. Close the **workspace** and then double-click the **app.callflow** for this project, new elements will appear under Local Folders (new Say it Smart plugins appear in the Say it Smart window)>

**VXML Server (Application Specific):**

1. Either manually or by using the Studio, you must deploy the Java classes to VXML Server into the directory
   iii. **VoiceXMLServer/application/<appname>/java/classes**
   iv. **VoiceXMLServer/application/<appname>/java/lib**

2. If a Java class in this location changes, you must run the application's **updateApp.bat** script. While this script is graceful for the application, it causes an immediate reload of Java classes. If this is not intended, suspend the application and wait until all current calls end.

## Global Deployment of Custom Java Components

1. **Call Studio:**
   A. Place components that are to be shared across all applications into
   **C:/Cisco/CallStudio/eclipse/plugins/com.audiumcorp.studio.library.common_xyz/**
   (where xyz is the most recent version number)
      - Use the **classes/** subdirectory for .class files
      - Use the **lib/** subdirectory for jar files.

   B. **NOTE** only the Java that creates new Elements or Say it Smarts is required to be located here.

   C. **RESTART** Studio and the new elements will appear in the element pane directly under the "Elements"

2. **VXML Server**
   A. Place components that are to be shared across all applications into one of the following:
      - **VoiceXMLServer/common/classes/**
      - **VoiceXMLServer/common/lib/**

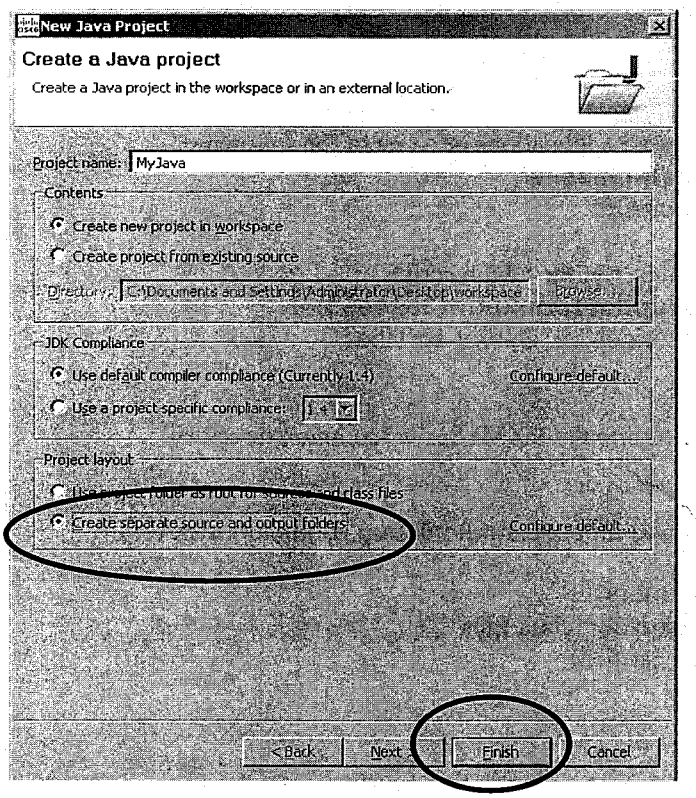
   B. If a java class in this location changes, you must run the admin script
      - **VoiceXMLServer/admin/updateCommonClasses.bat**
      - or restart the Cisco CVP VoiceXML Server service.

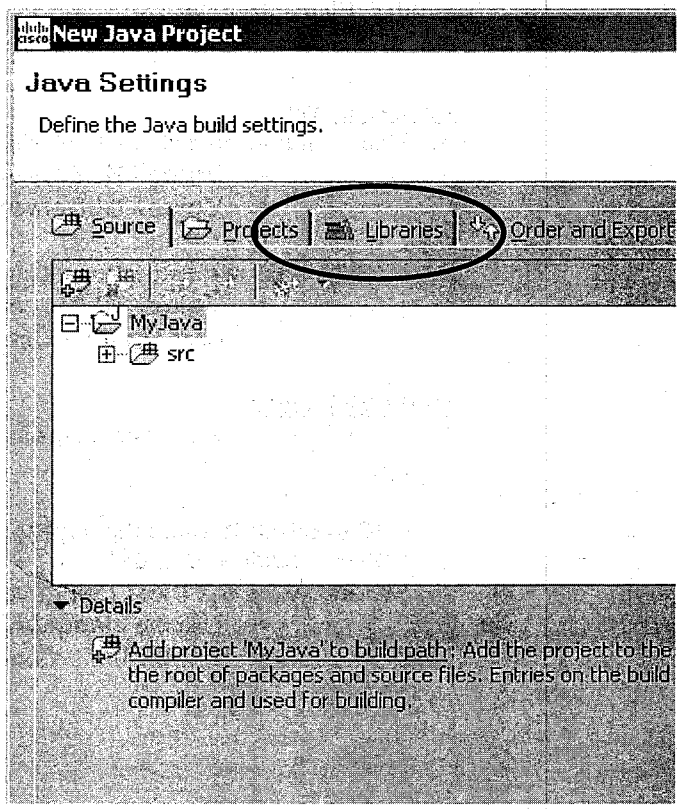
3. **Studio Debugger**
   A. Place the same Java components as you stored into the VXMLServer/common directory into

   **C:/Cisco/CallStudio/eclipse/plugins/com.audiumcorp.studio.debug.runtime_x.y.z/AUDIUM_HOME/common/**
   (where xyz is the most recent version number)
      - Use the **classes/** subdirectory for .class files
      - Use the **lib/** subdirectory for jar files

## Creating a Java Project in Eclipse (Reference)

1. Follow the menu path: **File > New > Project** and select to create a new "Java" project.
2. Click **NEXT >**
3. On the Create Java Project page
   - Enter a project name: **"myjava"**
   - Under **Project Layout:**
     click **Create separate source and output folders.**
4. Click **NEXT >**
5. Select the **LIBRARIES** tab

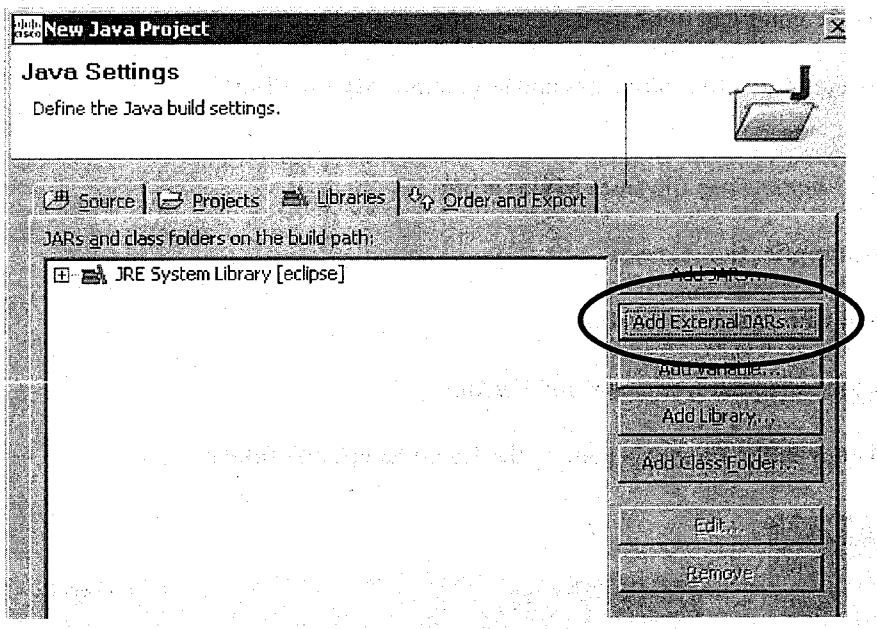


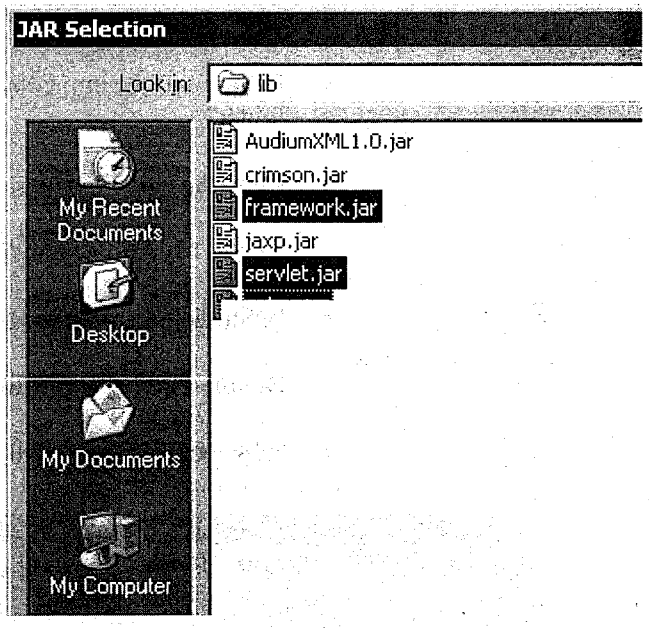



6. In the LIBRARIES tab,

**A.** Click **ADD EXTERNAL JARS,** add the following:

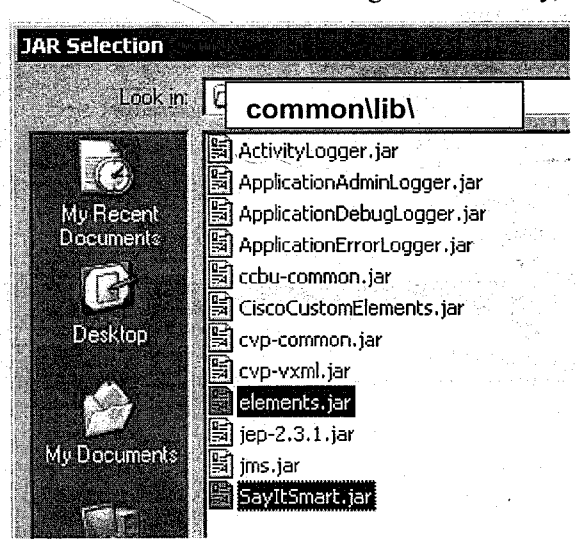
- Navigate to: **MyComputer: C: \ Cisco \ CVP \ VXMLServer\ lib**
- Holding the CTRL key, select **framework.jar, servlet.jar** - Then press **OPEN**

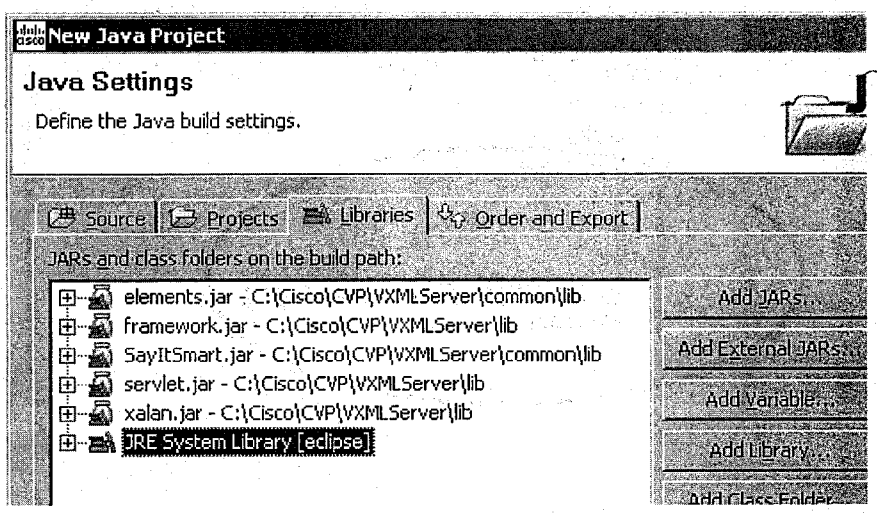



**B.** To extend existing Elements or Say it Smarts, click **ADD EXTERNAL JARS,** add the following:

- **Navigate to: C: \ Cisco \ CVP \ VXMLServer\ common\lib**
- Holding the CTRL key, select **elements.jar and SayItSmart.jar**




7. The setup is now complete. You will be asked if you'd like to switch to the Java Perspective. Press **NO.**

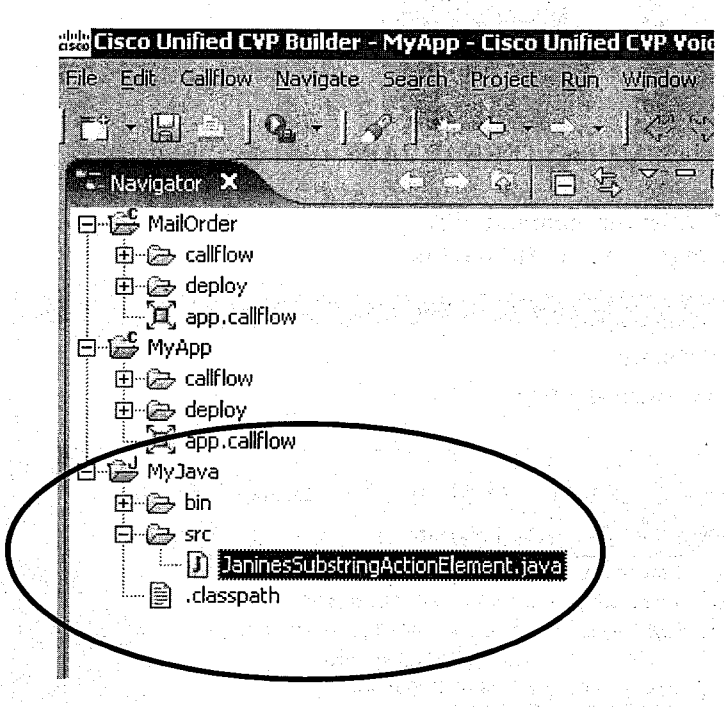
8. The instructor will provide you with prewritten Java components. To test a Java component, follow these steps:

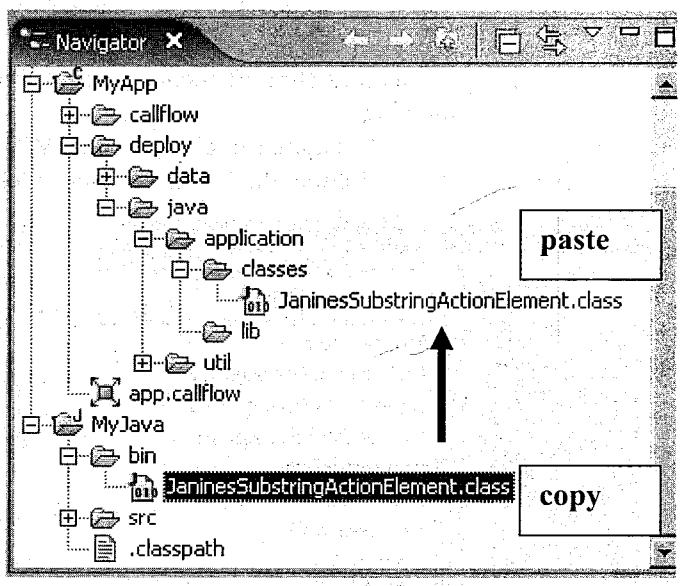
   - Copy and paste a file (***name*.java**) into Call Studio into the folder **MyJava/src/**

   This is the source code and must be compiled to be used.

   - Eclipse will be immediately compile into ***name*.class** and stored into **MyJava/bin/** folder.
   - Copy the compiled class file ***name*.class** file from **Myjava/bin**

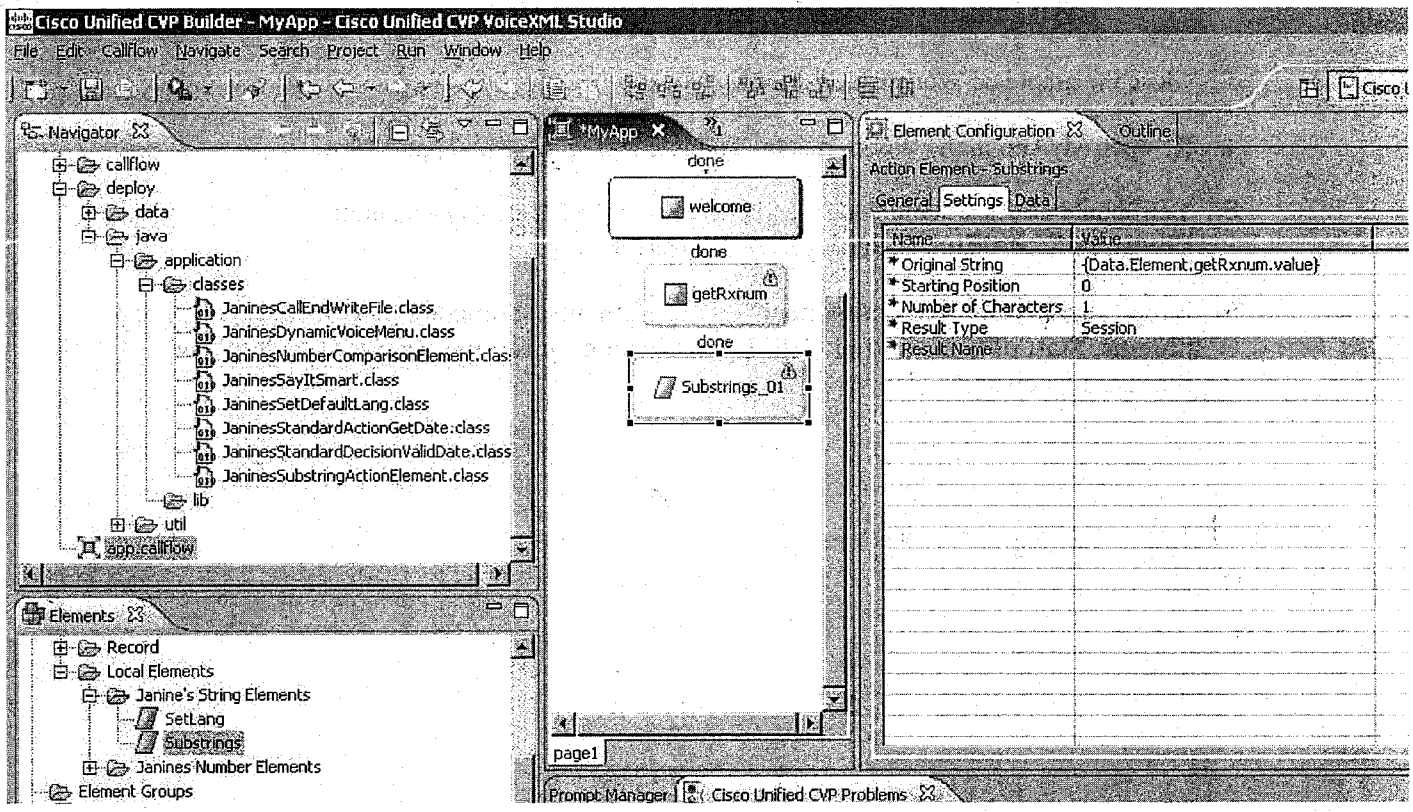
     Into **MyApp/deploy/java/application/classes/**
   - In Studio, close the graphical workspace of **MyApp**
   - Double-click **app.callflow** to reopen the workspace for MyApp
   - New local elements will display in the Elements pane in the **Local Elements/ folder**



9. If the custom component creates an element (Configurable Action, Decision or Voice element) then the new local elements will display in the **Elements pane** in the folder **Elements/LocalElements/**

10. Not all Java components create elements.



## Classroom Example

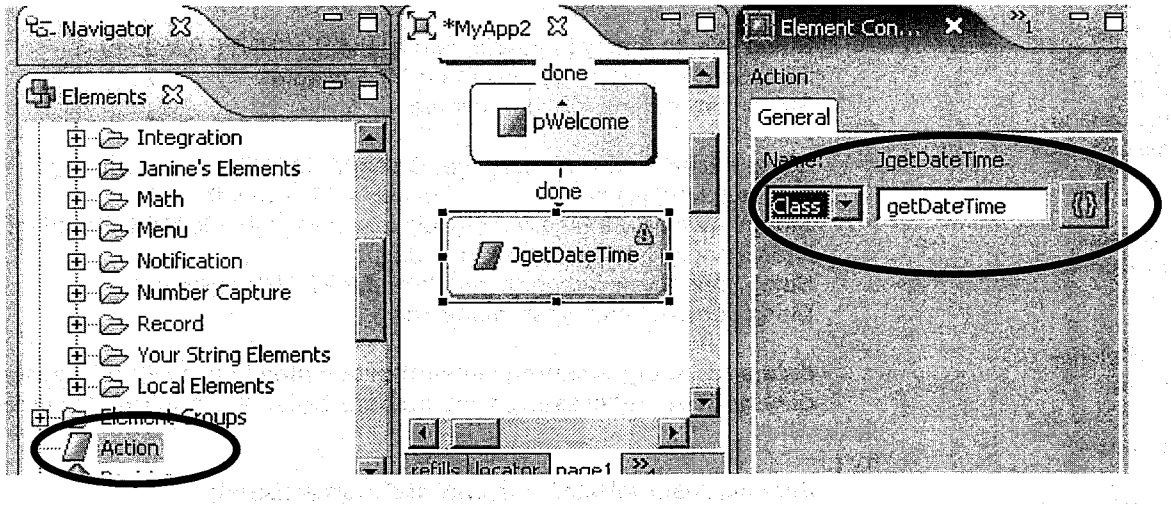
1. The examples below are in the **CVPD Course Files** folder on your PC.
2. Copy the precompiled Java classes
   - from **C:/CVPD Course Files/Janines Java Samples 2/ Java Classes / ***
   - into the **Studio Navigator** pane: **MyApp/deploy/java/application/classes**
3. Close the graphical Workspace Editor for MyApp.
4. Re-open MyApp in the Workspace by double-clicking MyApp/app.callflow
5. You should now see some folders and elements in the Elements pane, under **Elements/LocalElements/JaninesElements**
6. Recall that not all Java components create elements. Some are used for Say it Smart plugins, Start of Call, End of Call, Dynamic Configurations, Standard Actions, or Standard Decisions.

## Implementing Standard Actions

1. A **Standard Action** executes on the VXML Server and has one exit state named **done**. It is used for Java customization where variable names are determined ahead of time.

2. Standard Actions have no Settings tab, but can interface with VXML Server as follows:
   - Get Element and Session data values
   - Access Project Property settings
   - Create Element and Session data or reassign values to existing Session data
   - Add to the Activity log or Error log
   - Change the value of the Default Audio Path, Language, Session Timeout

3. To invoke a Standard Action, drag an Action element into the workspace. In the Configuration pane, select the source of the action code as Class (Java class) and enter the class name including the full package name (no file extension).
   - If there is no package, enter the class name:**getDateTime.**
   - If the java class is within a package (nested subdirectories), enter the full package name, such as **com.mycompany.getDateTime**




4. The **getDateTime** Java class (shown below) gets the date and time from the VXML Server operating system and stores them into Session data named **currentDate** (YYYYMMDD format) and **currentTime** (HHMM 24-Hour time format). It also logs them to the Activity log.

5. You can follow this with an Audio element and use Say It Smart to speak the date and time to the caller: **{Data.Session.currentDate}** or **{Data.Session.currentTime}.** Or you might use these variables for custom logging, etc.

## Standard Action Class (Reference Only)

1.Use a Standard Action element and set the Class Name to **getDateTime**

2. This java class contains Java code to get the date and time from the VXML Server operating system and stores them into **Session data** named **currentDate** (YYYYMMDD format) and **currentTime** (HHMM 24-hour format). This code also creates an Element variable named **successFlag** with the value **true** or **false** based upon whether it succeeds or fails.

Standard actions always start this way.
The public class name must match the file name

```
public class JaninesStandardActionGetDate extends ActionElementBase
{
```

This method **doAction** executes at runtime. It receives 2 parameters:

- *name* of the element executing
- *ActionElementData* (named *'data'* here) which is a version of the Session API, providing access to all runtime data and settings.

```
    public void doAction(String name, ActionElementData data) throws ElementException
    {
            Calendar c = Calendar.getInstance();
            c.setTime(new Date());
            String hour = new Integer(c.get(Calendar.HOUR_OF_DAY)).toString();
            if (hour.length() == 1) { hour = "0" + hour; }

            String minute = new Integer(c.get(Calendar.MINUTE)).toString();
            if (minute.length() == 1) { minute = "0" + minute;          }
            String currentTime = hour + minute;

            String month = new Integer(c.get(Calendar.MONTH)+1).toString();
            if (month.length() == 1) {          month = "0" + month;    }
            String day = new Integer(c.get(Calendar.DAY_OF_MONTH)).toString();
            if (day.length() == 1) { day = "0" + day;           }
            String year = new Integer(c.get(Calendar.YEAR)).toString();
            String currentDate = year+month+day;

            data.addToLog("creating session data called 'currentDate'",currentDate);
            data.addToLog("creating session data called 'currentTime'",currentTime);

        try {
            data.setSessionData("currentDate",currentDate);
            data.setSessionData("currentTime",currentTime);
            data.setElementData("successFlag","true", ActionElementData.PD_BOOLEAN
            ,true);
         } catch (Exception e) {
              data.setElementData("successFlag","false",ActionElementData.PD_BOOLEAN
                        ,true);
              e.printStackTrace();
            }
    }
}
```

*data.addToLog(label, value)* adds an entry to the activity log

*data.setSessionData(name, value)* creates and/or assigns a value to a session data variable

*data.setElementData(name, value,type, addToLogFlag)* creates and/or assigns a value to an element data variable for the current element

## Implementing Standard Decisions

1. A **Standard Decision** executes on the VXML Server and has multiple exit states.

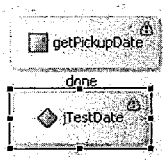
2. Standard Decisions have no Settings tab, but can interface with VXML Server:
   - Get Element and Session data values
   - Access Project Property settings
   - Create Element and Session data or reassign values to existing Session data
   - Add to the Activity log or Error log
   - Change the value of the Default Audio Path, Language, Session Timeout

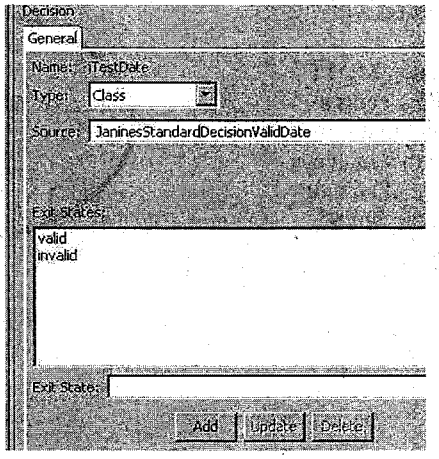
3. Decisions, unlike Actions, branch the code at runtime down an exit state that you **manually configure**. At runtime, if the Java class returns an exit state name that isn't configured, the VXML Server throws a Java exception.

4. For this example, use a **Date** element named **getPickupDate** to collect a date from the caller. Then use its Data tab to assign that value into Session data named **callerDate** (this is what the Java code accesses):

   Data tab: Name: **callerDate** (Case Sensitive)
   Value: **{Data.Element.getPickupDate.value}**
   Create: After

5. Use a **Decision** element and in the Configuration pane set the following:
   **Type**: Class
   **Source**: JaninesStandardDecisionValidDate
   - **Exit States:** Enter the name of each exit state and press **Add.** Caution - Case Sensitive!
     **Exit State: valid**
     **Exit State: invalid**




6A) In this example, if the caller says a full date (says *May third 2011* or DTMF: *YYYYMMDD*) the code branches down the **valid** path and you can speak the date to the caller using Say it Smart (YYYYMMDD).

6B) If the caller says *'today'*, or *'tomorrow'* or says a date without a year (*May third*), the code branches down the **invalid** path as this would cause a Say it Smart exception if it were spoken.

You can use an Audio element to tell the caller they must enter the full date and then loop back to the getPickupDate element.

## Standard Decision Class (Reference Only)

This code sample retrieves the value of a Session data variable named '**callerDate**' and checks if it can be spoken as YYYYMMDD format using the Date Say it Smart.

If the Session variable **callerDate** contains '**?**' or equals '0', '+1', or '-1' the code returns the Exit State '**invalid**' to indicate that the date received from the caller is not valid. Otherwise it returns the exit State '**valid**'. Your Studio application must configure exit states with these names.

All Standard Decisions must begin this way

```
public class JaninesStandardDecisionValidDate extends DecisionElementBase
{
   /**
    * Check if the session data 'callerDate' has any "?" or has the value
    * 0 (caller said 'today'), +1 ('tomorrow') or -1 ('yesterday')
    * return the exit state "valid" or "invalid"
    */
```

***doDecision*** is executed at runtime and receives 2 parameters:

A. *name* – element name executing this code
B. *DecisionElement Data* (called '*data*' here) – version of the Session API

```
public String doDecision(String name, DecisionElementData data) throws AudiumException
   {
            String callerDate = data.getSessionData("callerDate").toString();

            if (callerDate.indexOf("?")>=0 || callerDate.equals("0") ||
                        callerDate.equals("+1") ||  callerDate.equals("-1") ) {
               return "invalid";
            } else {
               return "valid";
            }
   }
}
```

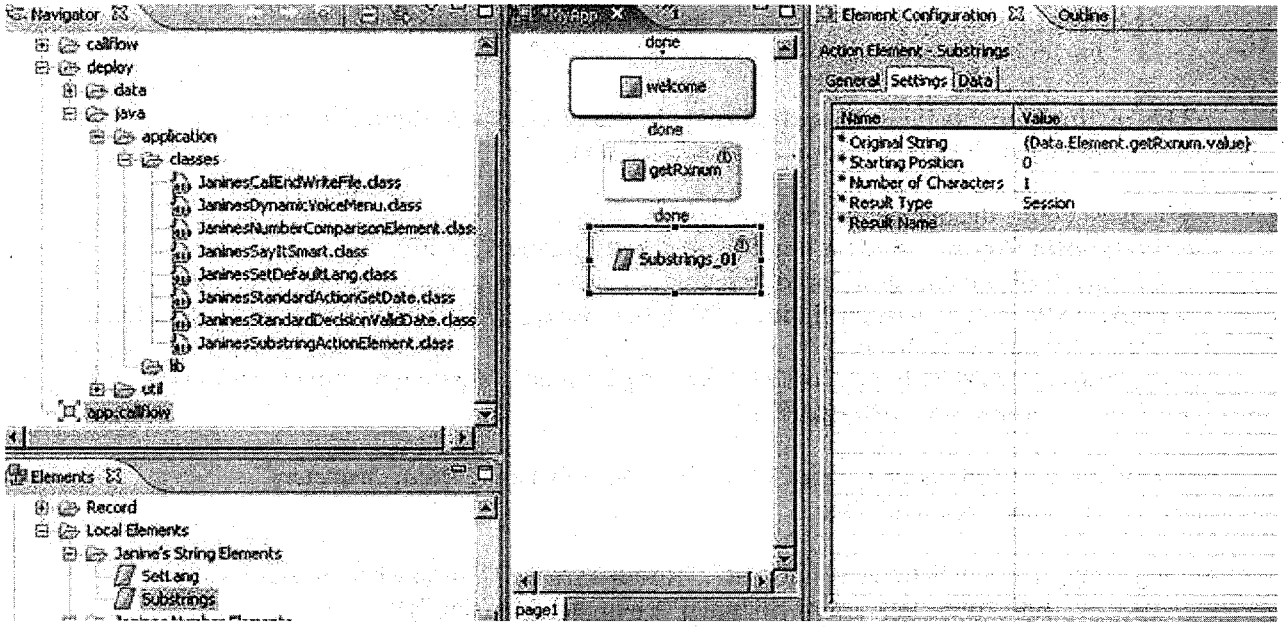
doDecision MUST end with a **return** statement that returns a string that matches the name of an exit state in the Studio application.

## Implementing Configurable Action and Decision Elements

1. Configurable Action and Decision Elements show up as elements in the Elements pane. They are the easiest and most flexible to use:
   - Self-documenting (description appears in the Elements pane, when hovering over the element)
   - Settings to allow you to pass data and specify where to receive data back
   - Settings can include descriptions, contain default values, be repeatable, allow substitution
   - Configurable Decision elements have the exit state names built-into them
   - You don't need to know the Java class name

2. Once the Java class or jar files are stored in the appropriate directory, and Studio is either restarted or the Workspace is closed and reopened, new elements will display in the element pane - either in the **Elements/Local Elements/** folder (if deployed in the application) or directly under the **Elements/** folder (if deployed globally).

3. Drag in the element to use it. The Settings tab may have required or optional configuration settings.



## Configurable Action Element Java Class (Reference Only)

This code sample creates an Action element named **Substrings**. It is included here purely for reference and is outside the scope of the CVPD course.

The Settings tab has five settings allowing you to pass in the original string, the start position and number of characters to retrieve. You may also specify where to store the desired output, into Element or Session data of any name you like.

```
public class JaninesSubstringActionElement extends ActionElementBase
implements ElementInterface
{
// This method returns the name of the action element that appears in Studio's Element view
public String getElementName()  { return "Substrings"; }

// This method returns the name of a folder that will contain the action element
public String getDisplayFolderName() {  return "Janine's String Elements"; }

// This method returns a description of the element that will display in Studio
public String getDescription() { return "This element returns a substring.\n" +
        "Into either element or session data as specified.\n";   }

//This method returns the settings to display in the element's configuration view
public Setting[] getSettings() throws ElementException
{
//You must define the number of settings here
Setting[] settingArray = new Setting[5];   //each setting must specify:
//real name, display name, description, required?,appears once?, substitution? type of entry
settingArray[0] = new Setting("input", "Original String","This is the original string", true,
true, true, Setting.STRING);
settingArray[1] = new Setting("startPos", "Starting Position", "Starting position (or offset) 0-
based", true,   true,   true,  Setting.INT);

settingArray[2] = new Setting("numChars", "Number of Chars", "Num characters.",
true,   true,   true,   Setting.INT);

settingArray[3] = new Setting("resultType", "Result Type",  "Choose where to store result",
            true  true,   false,  new String[]{"Element","Session"});

settingArray[4] = new Setting("resultName", "Result Name", Name of Data to Hold Result",
                    true,   // It is required
                    true,   // It appears only once
                    false,  // It does NOT allow substitution
                    Setting.STRING);
return settingArray;
}
```

**getElementData** returns an array of the element data variables this element creates. This information appears in the Substitution Tag builder when someone selects a Substsring element.

***doAction*** is executed by VXML Server at runtime.

It starts by retrieving the Configuration (***config***) and then retrieves each ***SettingValue*** by name.

Each Setting is returned as a String and must be converted into the appropriate type.

Use ***try/catch*** blocks to provide a graceful exit if an error occurs.

In this case, the element creates element data named 'status' with the value 'failure' if an error occurs.

```
// If this element creates element data, specify it here.
public ElementData[] getElementData() throws ElementException
  {
   ElementData[] elementDataArray = new ElementData[1];
   elementDataArray[0] = new ElementData("myData", "My Element Data");
   return elementDataArray;
  }

//This is the run time code executed by VXML Server
public void  doAction(String name, ActionElementData actionData) throws AudiumException
 {
    try {
          // Get the configuration
          ActionElementConfig config = actionData.getActionElementConfig();

         // retrieve each setting value using its 'real' name as defined in the getSettings method above
         //each setting is returned as a String type, but can be converted.
         String input =  config.getSettingValue("input", actionData);

         // get the setting value for each setting's real name
         int startPos =    Integer.valueOf( config.getSettingValue("startPos",actionData)).intValue();
         int numChars = Integer.valueOf( config.getSettingValue("numChars",actionData)).intValue()
         String resultType = config.getSettingValue("resultType",actionData);
         String resultName = config.getSettingValue("resultName",actionData);

         //get the substring
         String sub = input.substring(startPos,startPos+numChars);

         //Now store the substring into either Element or Session data as requested
         //and store it into the variable name requested by the Studio developer
         If (resultType.equals("Element")) {
               actionData.setElementData(resultName,sub);
         } else {
               actionData.setSessionData(resultName,sub);
         }
        actionData.setElementData("status", "success");
      } catch (Exception e) {
               //If anything goes wrong, create Element data 'status' with the value 'failure'
               //and return an empty string into the variable requested by the caller
               e.printStackTrace();
               actionData.setElementData("status","failure");
        }
  }
}
```

## Configurable Decision Element Java Class (Reference Only)

All configurable Decision elements begin this way.

```
public class JaninesNumberComparisonElement extends DecisionElementBase implements
ElementInterface
{
```

Similar to Standard Decision elements, but have many more methods which are used by Studio.

```
  // This is the name of the decision element which appears in Studio.
  public String getElementName()   {   return "NumberCompare"; }
 //Folder name containing the decision element in Studio
  public String getDisplayFolderName() { return "Janine's Number Elements"; }
  // Description of the decision element in Studio
  public String getDescription()  { return "Compare 2 numbers.";   }
   //The Settings for the decision element in Studio
  public Setting[] getSettings() throws ElementException
  {
     Setting[] settingArray = new Setting[3];
     settingArray[0] = new Setting("num1", "First Number",
                                   "This is the first number in the comparison.",
                                   true,   // It is required
                                   true,   // It appears only once
                                   true,  // It does allow substitution
                                   Setting.FLOAT);
     settingArray[1] = new Setting("op", "Comparison Operator",
                                  "description", true, true, false, new String[] {"<", "<=", ">", ">=", "==", "!=","null"} );
     settingArray[2] = new Setting("num2", "Second Number",
                                "2nd number to compare", true, true, true, Setting.FLOAT);
    return settingArray;
  }
```

Methods starting with 'get' are used by Studio to display the element

A. getElementName
B. getDisplayFolderName
C. getDescription
D. getSettings
E. getElementData
**F. getExitStates**

Settings are the same as for Action elements

**getExitStates** returns an array of Exit States that display in Studio.

Always use the same real and display name for Exit States.

```
  public ExitState[] getExitStates() throws ElementException
  {
     ExitState[] exitStateArray = new ExitState[3];
     exitStateArray[0] = new ExitState("true", "true", "");
     exitStateArray[1] = new ExitState("false", "false","");
     exitStateArray[2] = new ExitState("error", "error","");
     return exitStateArray;
  }
public String doDecision(String name, DecisionElementData decisionData) throws ElementException
  {     // Get the configuration
        DecisionElementConfig config = decisionData.getDecisionElementConfig();
        try {
           float num1 = Float.valueOf(config.getSettingValue("num1", decisionData)).floatValue();
           String op = config.getSettingValue("op", decisionData);
           float num2 = Float.parseFloat(config.getSettingValue("num2", decisionData)).floatValue();
           //return the string 'true' or 'false'
           if(op.equals("<")){     if(num1 < num2) return "true"; else return "false";       }
           if(op.equals("<=")){    if(num1 <= num2) return "true"; else return "false";      }
           if(op.equals(">")){     if(num1 > num2)  return "true"; else return "false";      }
        } catch (RuntimeException e) {
        return "error";
      } return "error"; } }
```

The doDecision MUST end with a **return** statement that **returns the REAL name of an exit state** defined in the getExitStates method above.

## Endpoint Settings

These settings require custom java components for added functionality when the application is loaded/unloaded from memory (On Application Start/End) or when each caller Enters/Exits the application (On Call Start/End).

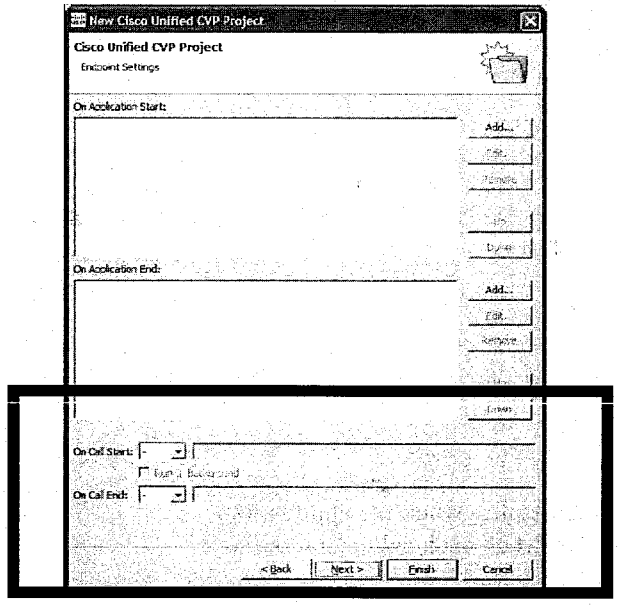
1. **On Application Start.** The On Application Start section specifies the Java classes that are executed when the voice application is started on the VXML server (boot time or deployApp or updateApp).
   - Useful for creating Global (persistent) variables used by all calls.

2. **On Application End.** The On Application End section specifies the Java classes that are executed when the voice application is stopped on the VXML Server. (shut down or releaseApp or suspendApp).
   - Useful for deleting Global (persistent) variables.

3. **On Call Start.** This specifies a Java class or URI to be accessed when the application is entered by a caller.
   - Useful for creating Session variables based upon ICM-passed data, contents of a properties file, database retrieval.

4. **On Call End.** This specifies a Java class or URI to be accessed no matter how the application visit for a caller ends: caller hang up, return to ICM, an unrecoverable error, application transfer to another Studio application, etc.
   - Useful for creating customer detail records or reporting based upon callflow activity.

## Start and End of Call




1. The **On Call Start and On Call End** classes are invoked through the Project Properties window of Call Studio:

**Project / Properties/ CallStudio / Endpoint Settings**

2. Select either **On Call Start** and/or **On Call End**
   A. Enter the Java class name (no file extension)
   B. **On Call Start** classes can run in the foreground or Background and are often used to create Session data variables used by the application. They can also set properties in the root document.
   C. **On Call End** actions execute **any time** the visit to this application ends (application transfer, hangup, error, etc). The **On Call End** has access to all data about the call, as well as how the call ended (hangup, disconnect, application_transfer, call_transfer, app_session_complete) and the call result (normal, max_ports, suspended, error, timeout, invalidated)

**Classroom Example:**

1. Edit the On Call Start. Select **Class**. Enter **JaninesCallStart**.
2. This java class reads the contents of a file into Session data it creates named **mod** (message of the day), which you can then speak to the caller.
3. In the Studio application, use an Audio element after the greeting and speak as TTS the value of the Session variable created by this java class:
   **The message for today is {Data.Session.mod}**
4. Save and Deploy and Update the application.
5. Now create the file:
   **C:\Cisco\CVP\VXMLServer\applications\MyApp\data\misc\mod.txt**
6. Edit and save the **mod.txt** so it contains a message of the day (e.g., *We have a special on vitamins this week. Two for the price of one*). Call in to hear the message. If you edit the mod.txt and save it, call in again to hear the new message. No updateApp required.

## On Call Start Class (Reference Only)

This code sample creates an On Call Start action that reads the contents of a file containing a message of the day into a Session data variable named 'mod'.

All OnCallStart actions begin this way

The ***onStartCall*** is the actual action that VXML runs.

The ***CallStartAPI,*** passed as a parameter, is a version of the Session API

**callStartAPI.addToLog** adds information to the Activity Log.

**callStartAPI.getApplicationDirectory** returns the full directory path where this application is executing at runtime

This portion reads the contents of the file into a Java variable name 'mod'

If the file wasn't found or couldn't be read, this code catches the exception and continues.

This stores the value of the Java variable 'mod' into Session data variable named 'mod'

```
public class JaninesCallStart implements StartCallInterface
{
   /** All On Start Call classes must implement this method.  This method will
    * be called at the beginning of the visit to the app.  Note that this method runs in a
    *separate thread from the rest of the call, so there is no guarantee that this method will
    *complete its execution before parts of call flow are reached.
    **/
   public void onStartCall(CallStartAPI callStartAPI) throws AudiumException
   {
        callStartAPI.addToLog("*******","entered Janines Start Action");

      // The file should be <Application Folder>/data/misc/mod.txt
      String fileName = callStartAPI.getApplicationDirectory() +
                        File.separator + "data" +
                        File.separator + "misc" +
                        File.separator + "mod.txt";

      String mod = null;
      FileInputStream fileInput = null;

      try {
         // Open the file and read it.
         fileInput = new FileInputStream(fileName);

         byte[] fileContents = new byte[fileInput.available()];
         fileInput.read(fileContents);
         fileInput.close();
         mod = new String(fileContents);
           callStartAPI.addToLog("*******mod is",mod);

      } catch(FileNotFoundException e) {
         // If the file wasn't there, set the number of calls to 0.

      } catch (IOException ioe) {
         // If there was an I/O problem, set the number of calls to 0.

      }
      callStartAPI.setSessionData("mod", mod);
   }
}
```

DATA.ADDTOLOG("****", Hi MOM!")





JAVA PROJECT

DEPOIS DE ALTERAR DE ACTUALIZA O SOURCE COPIADO PAR AS CLASSE DENTRO DO PROJETO E RECOPILAR

## **Call End Class** (Reference Only)

This code sample creates an On Call End action that appends the value of 2 Session data variables ('rxnum' and 'rxprice') to a customer detail record file that already exists.

All OnCallEnd actions begin this way

The ***onEndCall*** is the actual action that VXML runs.

The ***CallEndAPI*** passed as a parameter *(named endpointAPI here)* is a version of the Session API

**callEndAPI.getApplicationDirectory** returns the full directory path where this application is executing at runtime

Attempt to retrieve the Session data variables named 'rxnum' and 'rxprice' and appends information to the Java string variable **logstring**.

This portion attempts to open and append the *logstring* to the file.

If the file wasn't found or couldn't be written to, this code catches the exception and continues.

If the 'rxnum' or 'rxprice' Session variables didn't exist, this code catches the exception and continues.

```
public class JaninesCallEndWriteFile implements EndCallInterface
{
  /**
   * All On End Call classes must implement this method.  This method will
   * be called at the end of the call, as soon as the call flow has been completed.
   */
  public void onEndCall(CallEndAPI endpointAPI) throws AudiumException
  {
   // The file should be <Application Folder>/data/misc/cdr.txt
    String fileName = endpointAPI.getApplicationDirectory() +
    File.separator + "data" + File.separator + "misc" +File.separator + "cdr.txt";

    String logstring="";
    try {
       //Try to get rxnum and rxprice from Session Data.
      String rxnum = endpointAPI.getSessionData("rxnum").toString();
      String rxprice = endpointAPI.getSessionData("rxprice").toString();
      logstring = "rx number=" + rxnum + ",rx price=" + rxprice + "\r\n";
      FileOutputStream fileOutput = null;
       try {
        // Open the file and add to it.
        fileOutput = new FileOutputStream(fileName, true);
        byte[] fileContents = logstring.getBytes();
        fileOutput.write(fileContents);
        fileOutput.close();

        endpointAPI.addToLog("***On Call End","Added rxnum, rxprice to cdr.txt");
        } catch(FileNotFoundException e) {

          // If the file wasn't there, log it to the Activity Log
          endpointAPI.addToLog("***On Call End","Could not find file cdr.txt");

        } catch (IOException ioe) {
          // If there was an I/O problem,  log it to the Activity Log

          endpointAPI.addToLog("***On Call End","Error updating file cdr.txt");
        }
       } catch (Exception e1) {
      //If either rxnum or rxprice isn't in Session data, log a note to the Activity Log.
      endpointAPI.addToLog("***On Call End","Could not retrieve rxnum,rxprice");
    }
  }
}
```

## Dynamic Configurations

Dynamic Configurations allow Java code to configure the Settings and/orAudio for any element at run time. This is useful for creating dynamic menus based upon a caller's account number because the Java code can retrieve Element and Session data, use that to query a database for the audio file names, associated key presses or spoken utterances, language, and audio path to use.

### Classroom example:

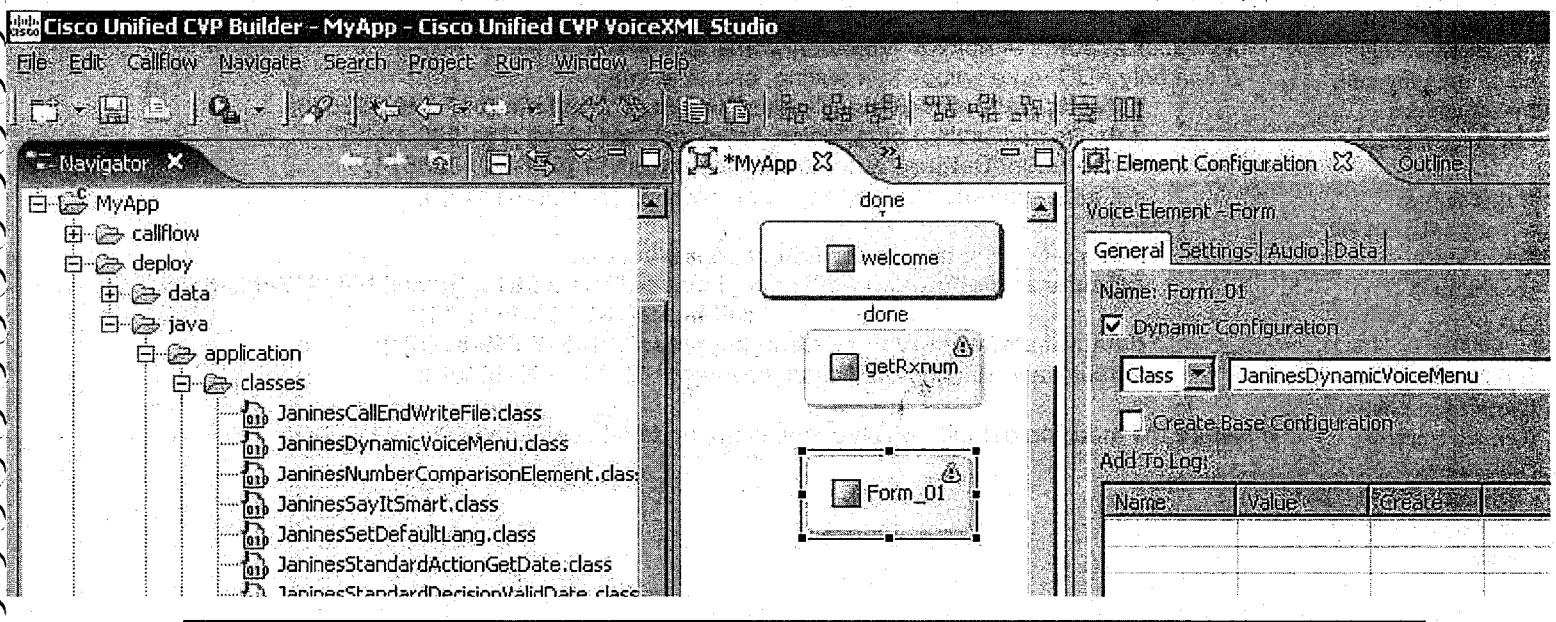
1. Use a Form element and name it **getInsuranceType**.

- Select the **General tab** of the element. Select the **Dynamic Configuration** checkbox
- Enter the java class name **JaninesDynamicVoiceMenu**
- You may select the **Base Configuration** checkbox if you'd like the Studio Settings and Audio to be passed to the java to use, modify, or overwrite.
- **Debugging Tip**: If you do not pass a Base Configuration, then the Dynamic Configuration MUST set all the required settings (marked with * in the Settings tab)

2. This example, creates an insurance menu by configuring the audio and allowed key presses. In this classroom simulation, the selections are the same for all caller's.

3. VXMLServer then executes the Form element and stores the caller's selection into element data named **value**.

4. Follow this with an Audio element, named pInsuranceType that speaks:
**You selected {Data.Element.getInsuranceType.value}**



## Dynamic Configuration for a Form Element (Reference Only)

This code sample creates a dynamic configuration for a Form element to provide a dynamic menu. The code configures voice keywords, dtmf keypresses, and an initial prompt.

```
public class JaninesDynamicVoiceMenu implements VoiceElementInterface
{

    public VoiceElementConfig getConfig(String name, ElementAPI elementAPI, VoiceElementConfig defaults)
                                                                                    throws AudiumException
    {
        //set up the Audio prompts for the initial audio
        VoiceElementConfig.AudioGroup initialAudioGroup;

        //If no audio exists yet, then create an initial audio group. Else add to existing.
        if (defaults.getAudioGroup("initial_audio_group",1) == null) {
            initialAudioGroup = defaults.new AudioGroup("initial_audio_group", true);  //true=allow bargein
        } else {
            initialAudioGroup = defaults.getAudioGroup("initial_audio_group",1);
        }
        VoiceElementConfig.StaticAudio prompt1 = defaults.new StaticAudio("Say call bill or press 1.", null);
        VoiceElementConfig.StaticAudio prompt2 =  defaults.new StaticAudio("or call home or press 2.", null);

        initialAudioGroup.addAudioItem(prompt1);
        initialAudioGroup.addAudioItem(prompt2);

        // Add the initial Audio Group to the Voice Element Configuration.
        defaults.setAudioGroup(initialAudioGroup);

        //Now work on the voice keyword settings to build the grammar
        String[] VoiceKeywords = defaults.getSettingValues("voice_keyword",elementAPI);

        if (VoiceKeywords != null &&  VoiceKeywords.length!=0){
        // can only ADD more values to a setting, one at a time
            defaults.addSettingValue("voice_keyword","call bill [617-549-8585]");
            defaults.addSettingValue("voice_keyword","call home [781-990-1040]");
            defaults.addSettingValue("dtmf_keypress","1 [617-549-8585]");
            defaults.addSettingValue("dtmf_keypress","2 [781-990-1040]");
        } else {
            //create NEW setting, in 2 different possible ways.
            defaults.setSettingValues("voice_keyword",new String[]{"call bill [617-548-8580]",
                                                    "call home[781-990-1111]" });
            defaults.setSettingValue("dtmf_keypress","1 [617-549-8585]");
            defaults.addSettingValue("dtmf_keypress","2 [781-990-1040]");
        }
    // Return the modified defaults which contains the updated configuration return defaults;
    }
}
```

getConfig creates the configuration

*ElementAPI* is the Session API.

*VoiceElementConfig* (named *defaults* here) is the configuration.—It may contain the Base Config. It must be returned at the end of this code.

StatisAudio is URI/TTS (as opposed to Say it Smart)

This configures the settings. The real names of settings are in the Element Specifications reference manual.

If this setting already exists in the Base Config, then we add to it.

If this setting doesn't yet exists then we create it (this shows 2 different ways to create a new setting value).

## Say it Smart Plugins

1. Custom Say it Smart Plugins are necessary for foreign language parsing of dates, currency, numbers; as well as for converting other data into audio file names.

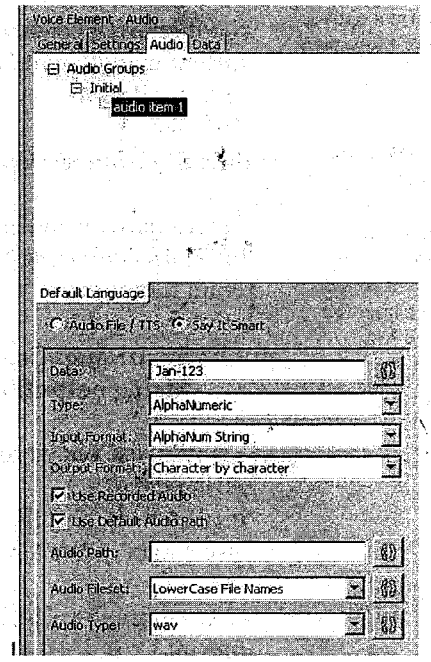
2. Custom Say it Smart Plugins display in alphabetical order along with the default plugins within the Audio tab of the any Voice Element.

3. **Classroom Example**

   A. Studio does not contain a Say it Smart plugins to spell items to callers. But you can use the custom **Alphanum** Say it Smart for this. The **Alphanum** Say it Smart converts the contents of the Data field into audio file names (and back up TTS). It ignores anything other than alphanumerics.

   B. For example the Alphanum Say it Smart would convert **JAN-123@** into this list of audio files with back-up TTS being sent to the gateway at runtime:

**<audio src="http://10.1.78.72/en-us/sys/j.wav> j </audio>**
**<audio src="http://10.1.78.72/en-us/sys/a.wav> a </audio>**
**<audio src="http://10.1.78.72/en-us/sys/n.wav> n </audio>**
**<audio src="http://10.1.78.72/en-us/sys/1.wav> 1 </audio>**
**<audio src="http://10.1.78.72/en-us/sys/2.wav> 2 </audio>**
**<audio src="http://10.1.78.72/en-us/sys/3.wav> 3 </audio>**




4. Select the **Alphanum** Say it Smart
   - Enter something in the data field (enter text or use the contents of a variable)
   - Select **"Use Recorded Audio"** (optional)
   - Select **Use the Default Audio Path** (if you've configured one).
   - Select Fileset: **Lower Case File Names** (CVP installs wav files with lower case file names)
5. Save, Deploy, Update, and test the application.

6. To see the audio file names sent to the gateway test using the Studio Debugger.

## Say it Smart Plugin Class (Reference Only)

This java code creates a Say it Smart plugin named Alphanum discussed on the previous page.

All Say it Smart plugins start this way

```
public class JaninesAlphanumSayItSmart extends SayItSmartBase implements SayItSmartPlugin
{
```

***getDisplayInformation*** is used by Studio. You define:
A. the name of the plugin
B. all Input formats
C. all Output formats
D. all Filesets

```
    public SayItSmartDisplay getDisplayInformation() throws SayItSmartException
    {
        SayItSmartDisplay toReturn = new SayItSmartDisplay("alphanum", "AlphaNumeric", "");
        toReturn.addInputFormat("input", "AlphaNum String", "");
        toReturn.addOutputFormat("output","Character by character","");
        toReturn.addFileset("filesetUpper","UpperCase File Names","");
        toReturn.addFileset("filesetLower","LowerCase File Names","");
        return toReturn;
    }
```

getFormatDependencies – for each Input format above, define the list of Outputs to display

```
    public SayItSmartDependency getFormatDependencies() throws SayItSmartException
    {
      SayItSmartDependency toReturn = new SayItSmartDependency();
      toReturn.addParent("input", new String[] {"output"});
      return toReturn;
    }
```

getFilesetDependencies – for each Output format above, define the list of Filesets to display

```
    public SayItSmartDependency getFilesetDependencies() throws SayItSmartException
    {
      SayItSmartDependency toReturn = new SayItSmartDependency();
      toReturn.addParent("output",new String[]{"filesetUpper","filesetLower"});
      return toReturn;
    }
```

***convertToFiles*** is used by VXML Server at runtime.

It uses the ***toReturn.add*** to add each audio file name, alternate TTS text, and a boolean flag whether the TTS contains SSML markup.

```
    public SayItSmartContent convertToFiles(Object dataAsObject, String inputFormat,
          String outputFormat, String fileset) throws SayItSmartException
    {
      SayItSmartContent toReturn = new SayItSmartContent();
      String data;
      if (fileset.equals("filesetUpper")){ data = dataAsObject.toString().toUpperCase();   }
      else { data = dataAsObject.toString().toLowerCase();  }

      /* Go through each character of the string and spell it.  */
      for (int i = 0; i < data.length(); i++) {
        char theChar = data.charAt(i);
        if(theChar>='0' && theChar<='9' || (theChar>='A' && theChar <='Z') || (theChar>='a' && theChar <='z') ){
          String theString = data.substring(i,i+1);
          toReturn.add(theString, theString+" ", false);
        }
      }
```

Return this back to VXML Server.

```
        return toReturn;
```

[End of Chapter ].

C:\Cisco\CVP\VXMLServer\MyAppl\Java\Application\Classes - Global

# Chapter 11

# Courtesy Callback

- Courtesy Callback Overview
- ICM Configuration and Demo Courtesy Callback Script
- Studio and VXML Server Courtesy Callback Applications
- Reporting Server Tables for Courtesy Callback
- Media Server Configuration for Courtesy Callback

ICM
① GOTO STUDIO APP
② QUENE CALC
③ OFFER CALLER A CALLBACK IN STUDIO APP.

## Courtesy Callback

1. **Courtesy Callback** is a standard feature of CVP 8.0(1) that reduces the time callers have to physically wait on hold in a queue. Callers who meet some criteria are offered the option to be called back by the system instead of waiting on the phone for an agent.

2. If the caller requests a callback:
   A. The caller **records their name** and **leave a phone number** which is stored in the CCB section of the Reporting Server database.

   B. A placeholder for their call remains in the queue and when an agent will soon be available (expected wait time is under 60s), the system places a call back to the caller.

   C. The caller answers the call and must confirm that they would like to accept the callback and the system connects the caller to the agent after a much shorter wait.

   - If the **callback is unsuccessful** (no answer, line busy, voice message machine answers), the caller's place remains in the queue, and the system attempts the callback again. The number of attempts and the delay between attempts are configured within the Studio application. If all tries are unsuccessful, the callback is removed from the database, the call is removed from the queue.
   - If the **callback is successful, but the caller does not accept the callback**, then the call is removed from the queue and marked as abandoned.

3. Courtesy Callback is executed via use of these downloads (located on the Call Server in the directory **C:\Cisco\CVP\OPSConsoleServer):**
   - One pre-written, customizable ICM Script
   - Five pre-written Studio/VXML Server applications (3 are customizable)
   - Gateway configuration using survivabililty.tcl must be configured
   - A CCB Real-time Callback database that is added to a fully licensed Reporting Server
   - CCB audio prompts to deploy to the Media Server

4. CCB uses a VXML Server license port the entire time the call is in queue.

## ICM Configuration

1. Create 2 network VRU Scripts –
   - VXML_Server_Interruptable (GS,Server,V,1) with Timeout > 7200
   - VXML_Server_Noninterruptable (GS,Server,V,2) with Timeout > 7200
   - Note the 4th parameter of GS,Server,V,# is any unique string
2. Create ToExtVXML array of size 5 x 60 (or more)
3. Create FromExtVXML array of size 4 x 60 (or more)
4. Create new ECC variable **user.CourtesyCallbackEnabled** (size 1) – this will be set in the ICM Routing script to indicate whether a call in the Queue can be offered callback
5. Add logic to the sample ICM Script to determine which callers will be offered CCB and merge this into your existing routing scripts at all points where calls are queued.

## Courtesy Callback Studio Applications

1. Unzip the Studio applications stored in
**C:\Cisco\CVP\OPSConsoleServer\StudioDownloads\CourtesyCallbackStudioScripts.zip**

2. Copy them into the Workspace Folder and import each Call Studio Project into Studio

3. In all five CCB applications, change the Default Audio Path under Project / Properties to point to your media server to the location of your CCB audio files (e.g., http://IP/en-us/app)

4. Do **NOT** customize these two Studio Applications other than to set the Default Audio Path:

   A. **CallbackEngine:** Keeps the VoIP leg of the call alive between the time the caller and hangs up and when the caller actually receives the callback. Do **not** modify this script.

   B. **CallbackQueue:** Handles the keepalive mechanism for the call when callers are in queue and listening to the music played by BillingQueue. Do **not** modify this script.

5. You **may** customize the following scripts to suit your business needs, and to modify the caller interaction portions, as well as to set the Default Audio Path.

   A. **CallbackEntry:** This application asks the caller if they'd like a callback and if so, it adds them to the CCB database. If the caller doesn't select a callback, then the BillingQueue application is repeatedly invoked using Subdialog_Invoke.

   B. **BillingQueue Application:** This plays an audio file of music to callers while they are in the queue. This application is repeatedly invoked to continually play short audio files.

   C. **CallbackWait Application:** When the caller is called back, this application plays their recording back to them and asks if they'd like to accept the callback. Modify the IVR treatment a caller receives when they are called back.

6. Once you've modified the Studio applications (see below) you must save and deploy them to all VXML Servers. Be sure to run the appropriate admin script (deployApp.bat or updateApp.bat) or to deploy using the Operations Console.

## Media Server Files For Courtesy Callback

1. **CCBAudioFiles.zip** has callback-specific application media files under **en-us\app** and media files for Say It Smart under **en-us\sys.** These special audio files should be unzipped and copied to your media server.

2. If you are using the VXML Server as a media server (e.g., a lab environment configuration), then copy all of the files to **VXMLServer\Tomcat\webapps\CVP\audio.**

3. The sample Studio Applicationss have the Default Audio Path set to **http://<server>:<port>/en-us/app** for the audio files. You will have to replace <server> and <port> with your media server IP address and port number.

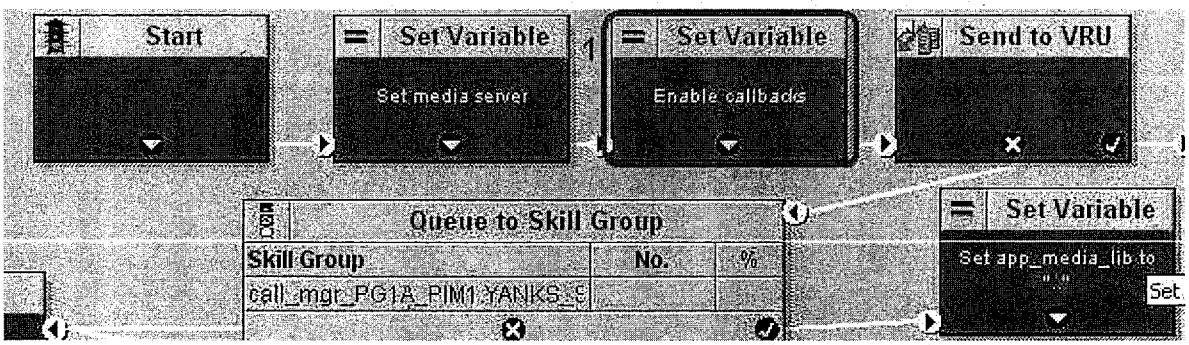
## Gateway Configuration

1. Gateway tcl scripts must be installed on the Ingress Gateway to enable CCB. For more details see the ***CVP Config and Admin Guide***.

## ICM Routing Script (CourtesyCallback.ICMS)

Once you've determined that the call must be queued to an agent, perform the following as shown in the sample **CourtesyCallback** ICM Routing script below. This **CourtesyCallback** download must be merged into your existing ICM scripts and modified as needed.

1. Decide whether this caller should be offered courtesy callback. If so, set **user.CourtesyCallbackEnabled** to **"1"** (true) and **Queue the call**.




2. Calculate **Expected Wait Time** for this skill group using a formula appropriate for your call center. Cisco provides this formula:

   (CallsInQueueNow *AvgQueueTime) /
   (#Agents)

```
Example,
(SkillGroup.%1%.RouterCallsQNow+1) *
(SkillGroup.%1%.AvgHandledCallsTimeTo5) /
 max( SkillGroup.%1%.Ready,
   (SkillGroup.%1%.TalkingIn +
   SkillGroup.%1%.TalkingOut  +
   SkillGroup.%1%.TalkingOther))
```

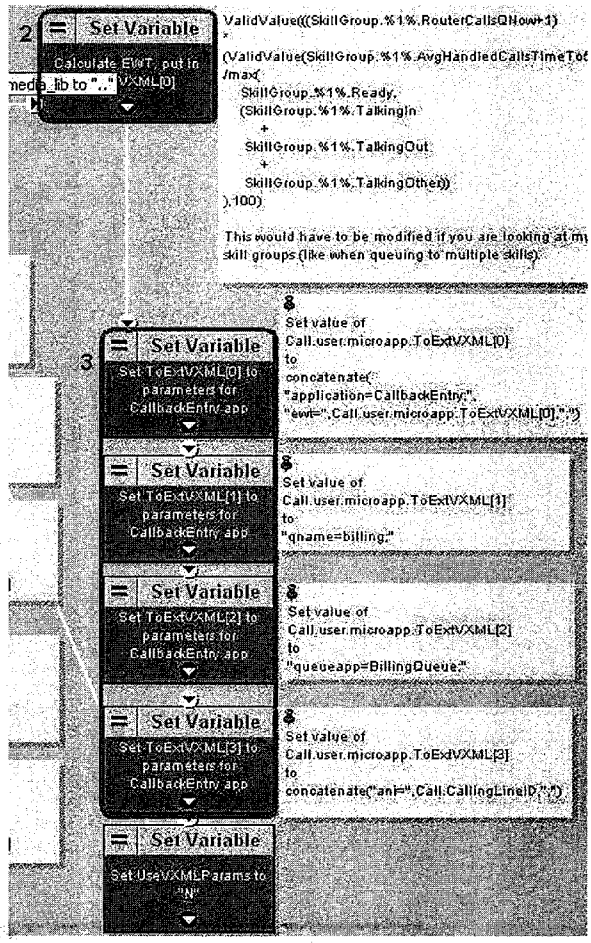
Note this is being assigned temporarily into ToExtVXML[0].

3. Assign into the ToExtVXML array the parameters to be passed to VxmlServer:

   **application=CallBackEntry;**
   **ewt=<*expectedWaitTime*>;**
   **qname=billing;**
   **queueapp=BillingQueue**

A) **Note** – You may modify the queue name **billing** to represent your queue.

B) In this case, you might also modify the name of the queueapp (**BillingQueue**) to be the name of a valid VXML Server application that will play music for the named queue. This is only necessary if you'd like different queues to hear different music.



5. The **RunExtScript node** invokes the Studio application **CallBackEntry** which offers the callback. If the caller does not accept the offer for a callback, **CallBackEntry** provides queue music until ICM interrupts and sends the call to an agent. (You may modify the caller interaction portion of the Studio app **CallBackEntry**).

## CallbackEntry Studio Application

This application asks if the caller wants a callback. If not, it plays queue music until an agent becomes available and ICM pulls the caller out. If the caller wants a callback, this app records caller's name and collects the caller's phone number as DTMF, adds the information into the CCB database, and returns back to ICM.

## A. CallbackEntry - Start Page

1. **Enter_Queue_01** - enters the call into the CCB database queue.

**The Callback_Enter_Queue** element is responsible for adding a new caller to the queue. This element must be executed for all callers even if the caller may not be offered a callback.

- Element Data: **ewt** – estimated wait time passed from ICM

2. **Ewt in Minutes** - converts the estimated wait time received from ICM to minutes

3. **Validate_01** – custom element that checks if the caller can be offered a callback based upon queue settings. Stores gateway information into element data.
   -If the caller can be offered a callback, then the exit state **'preemptive'** is followed.
   -If the caller can not be offered a callback (e.g., ewt is too short, max callbacks allowed has been reached, callback time is outside of allowed range), then the exit state **'none'** is followed.
   -If an **error** occurs, then the exit state **'error'** connects to the **Queue** page
   -If **refresh** exit state is followed, then execute the **SetQueueDefaults** node and try again to validate whether caller can be offered callback. If still unsure, connect to the **Queue** page.

   The **Callback_Validate** element is responsible for verifying whether or not a callback can be offered to the caller during this call. Depending on the outcome of the validation, the Validate element exits with one of four states.

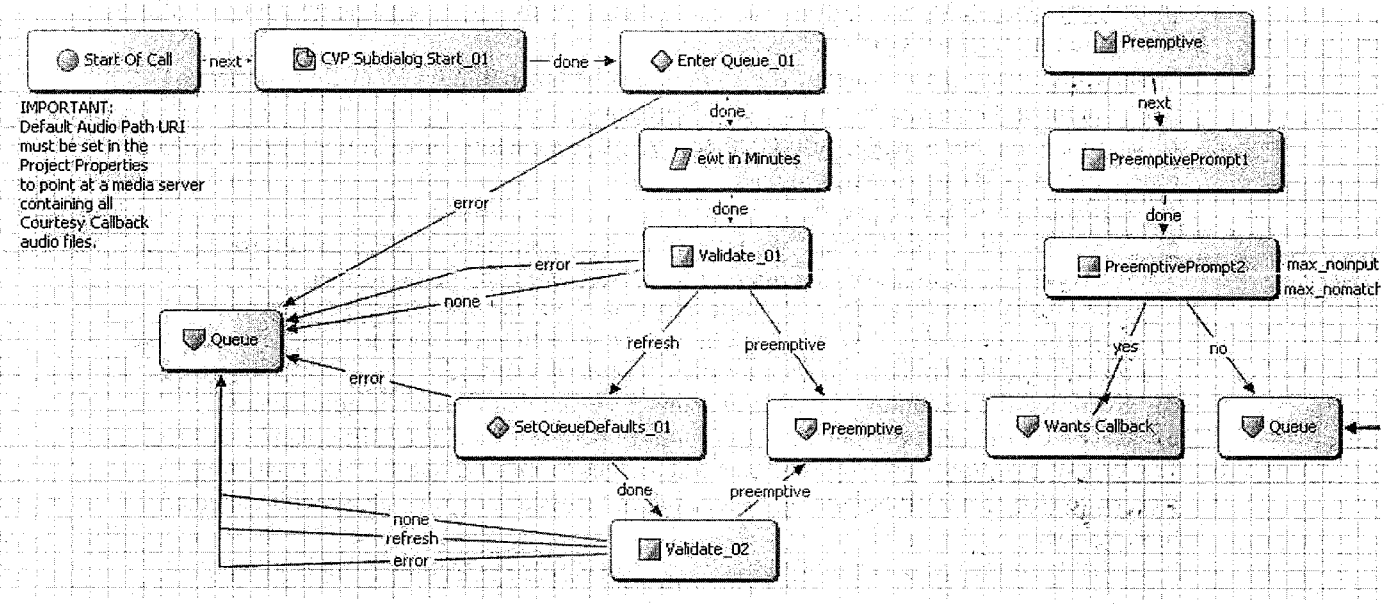
   A. Exit States:
      - **preemptive** – the callback is valid and can be offered during this call
      - **refresh** - The validation could not be performed because the DBServlet needs a reference data refresh. The app must call **SetQueueDefaults** before validation can occur.
      - **none** – callback not allowed
      - **error** – a value couldn't be retrieved

   B. Element Data:
      - **result** – exit state result
      - **ewt** – estimated wait time passed from ICM
      - **gw** – gateway identifier
      - **loc** – gateway location info
      - **capacity** – gateway capacity

4. **SetQueueDefaults -** this node can have different queue names. It is where you specify allowable callback times, max number calls allowed to be offered callback, etc for each queue.

## B. CallbackEntry - Preemptive Page Entry Element

1. **PreemptivePrompt1** (Audio element) plays the expected wait time to the caller;
2. **PreemptivePrompt2** (YesNo_Menu) asks the caller if they would like a callback.
   -If no, connect to the **Queue** page.
   -If yes, connect to the **Wants Callback** page.

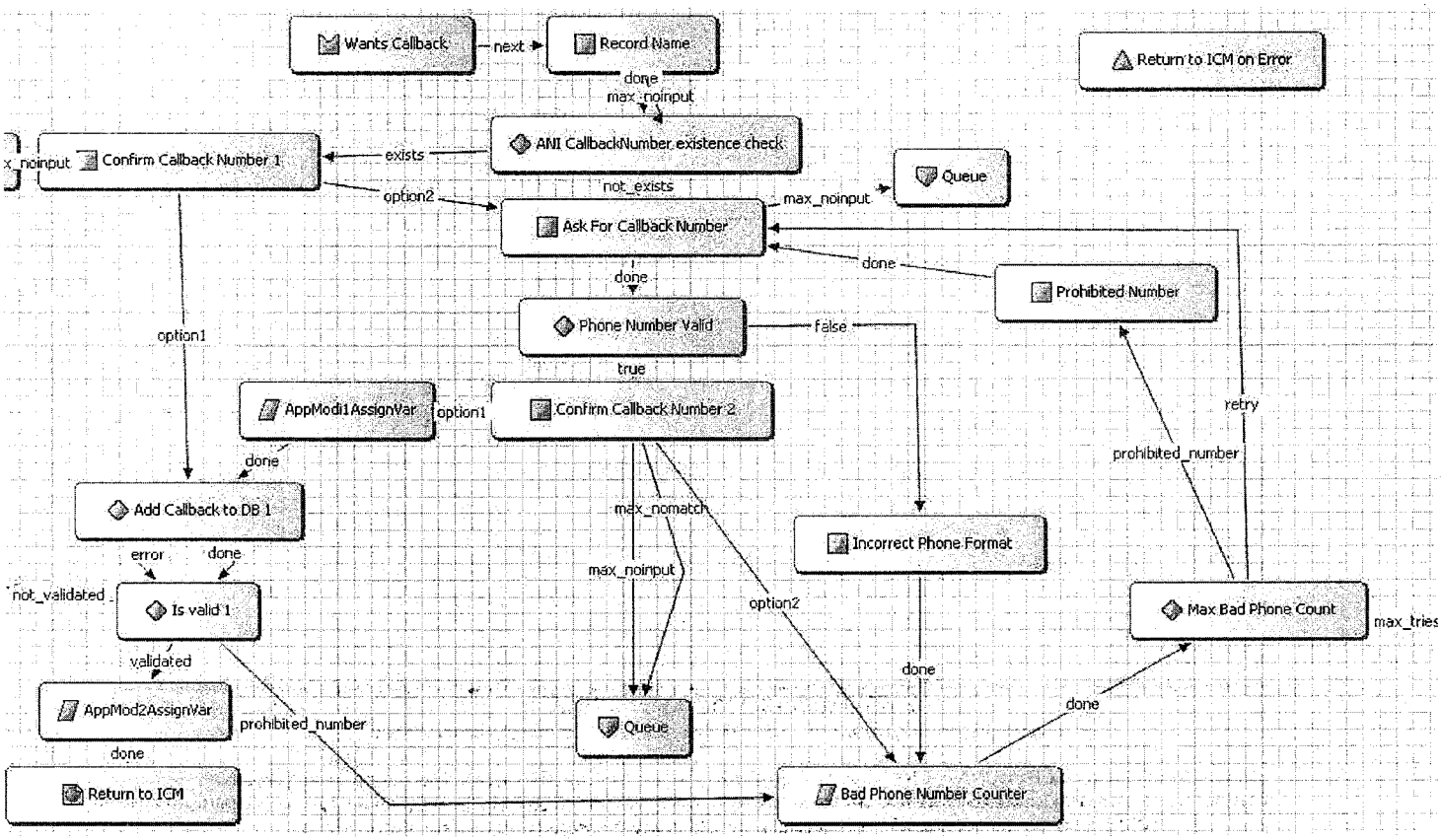


## C. CallbackEntry Application - Wants Callback Page

1. **Record Name** is a Record element that records the caller's name (played during callback).
2. **ANI CallbackNumber existence check** - Decision element -was ANI received from ICM?
3. **Ask For Callback Number** is a Digits element that collects the caller's phone number.
4. **Confirm Callback Number** is a 2_Option_Menu that confirms the caller's phone number or ANI (1 for correct, 2 for incorrect).
5. **Phone Number Valid** is a custom element that ensures the caller's phone number is 'valid' as configured through the Operations Console.

- If **not valid**, increment a counter, if the counter is less than 3, then try again to collect a phone number. Otherwise connect to the **Queue** page.
- If the phone number is **valid**, **Add Callback to DB** is a custom element that adds a callback object to the CCB DB. Settings include:
  a) Callback number (Required)
  b) Recorded Name File (Required)- URL to the recording above. Used by gateway to play the recording from the Media Server on callback. Must be in the format http://MediaServerIP/subdirectory/filename.wav or /CVP/audio/subdirectory/filename.wav
  c) Recorded Name Path - Windows path to the recording – (include this **only** if you'd like recording deleted). E.g., C:\Inetpub\wwwroot\en-us\app\rec\
  d) Recorded file retention – number of minutes before this file can be auto-deleted
  e) Recorded file deletion interval - # of minutes between checking if files can be deleted.

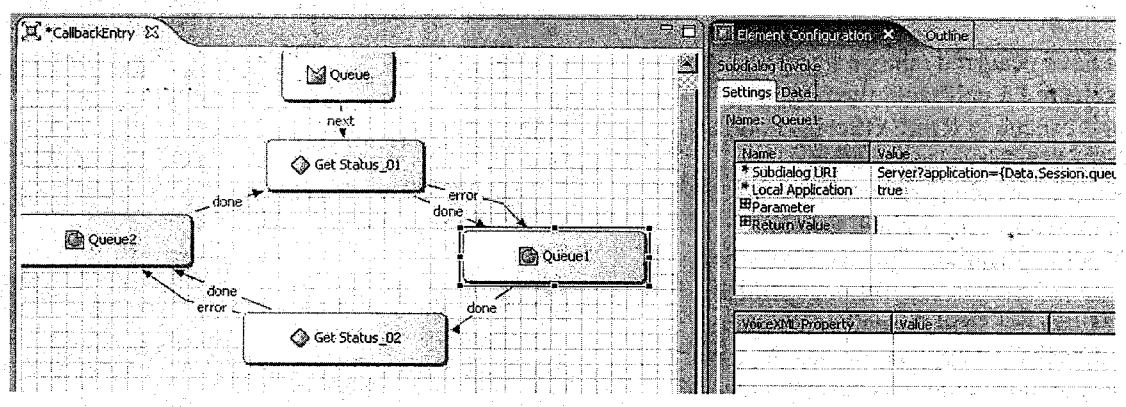
6. **AppModAssignVar** is an App Modifier that assigns SkipEndClass ← True (Data tab)
7. **Return to ICM** is a CVP Subdialog Return to send call back to ICM along with collected information.
8. Note that ICM then invokes the **CallbackEngine**/**CallbackWait** Studio applications.



10. On the **CallbackEntry-Queue** page, **Get Status_01** and **Get Status_02** are custom elements that retrieve all information about the callback related to this call. **Queue1** and **Queue2** perform a Subdialog Invoke to the application whose name was passed in from ICM as **queueapp** (eg, **BillingQueue**) that play a music audio file to the caller.

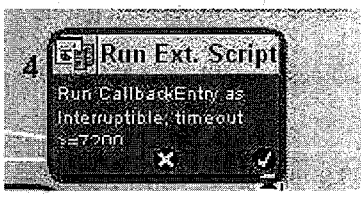
The **Get_Status_01 or Get_Status_02** element retrieves the following into Element Data:

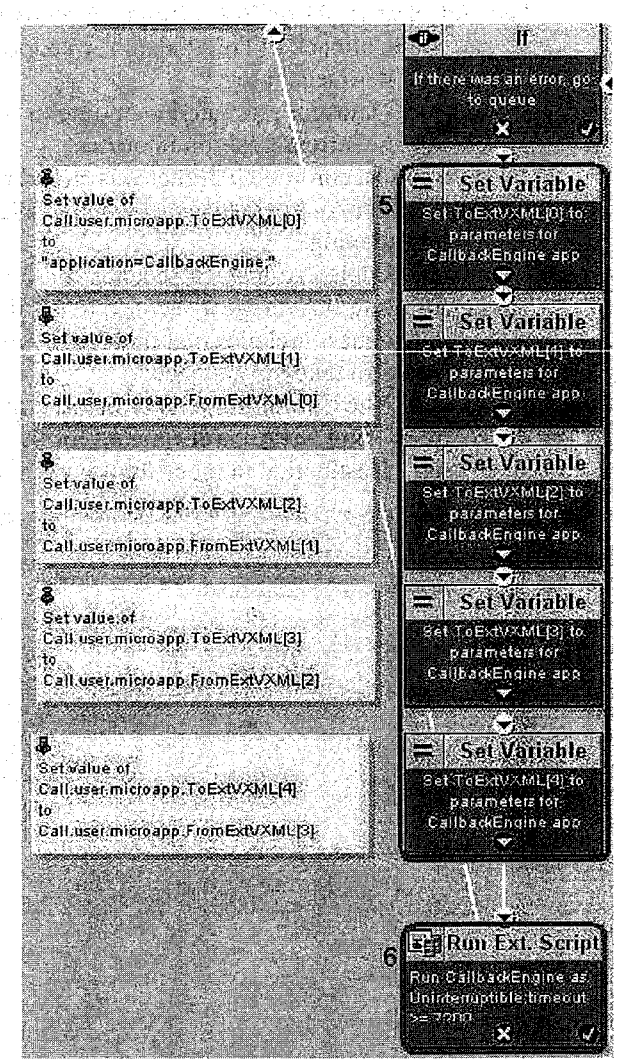
a) startCallback - (True/False) If true, perform callback based on caller position in queue and de-queue rate
b) ewt - Current estimated wait time in seconds before the callback should be initiated.
c) qpos- Current position in queue.
d) rec - Recording URL that was stored in the callback table.
e) DORateA- Average # seconds to leave this queue (includes going to agents and callers abandoning).
f) DORateB - Average # seconds for the #1 caller in this queue to leave the queue.
g) RORate - Average # seconds to get the caller back after starting the callback. Same for all queues. Includes dial time, ring time, and IVR time spent asking the caller if they are ready to take the callback.
h) cli - The Calling Line ID to be used for this callback
i) rna - Ring No Answer timeout for this call
j) dn - Destination number for this outbound call



## ICM Routing Script - Continued

If the caller selects a callback, then Callback Entry returns call flow back to ICM. ICM continues past the RunExtScript node.




5 If the caller has selected a callback, then **CallBackEntry** returns back to ICM. If the **CallBackEntry** returned an error, ICM sends the call to the Studio application **BillingQueue** to play music until an agent is available (not shown here).

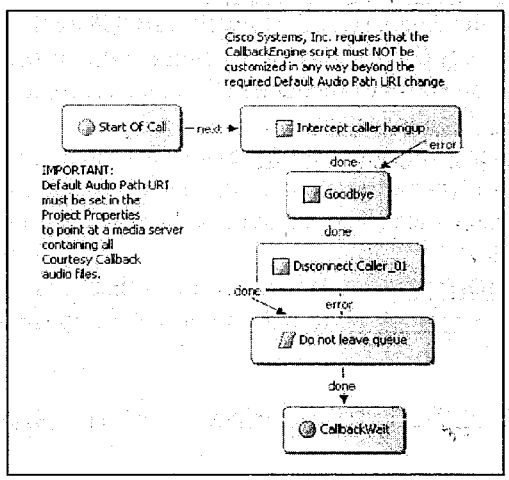
6 If no errors have occurred (shown here) and the caller **has selected a callback,** then set variables and execute a **RunExtScript** node to invoke the VXML Server application **CallbackEngine** which performs an application transfer to the VXML Server **CallbackWait** application. This RunExtScript node uses the GS,Server,V,**Uninterruptible.** That is, ICM can not send the call to an agent until the CallbackWait application has the caller on the phone, and the caller has accepted the callback.

A) These applications **CallbackEngine** and **CallbackWait** keep the call alive until the expected wait time is small.

B) When the wait time is small, the **CallbackWait** application instructs the system to dial the caller and ask if they'd like to **accept** or **cancel** the callback.

C) **CallbackWait** then returns the caller's decision back to ICM.

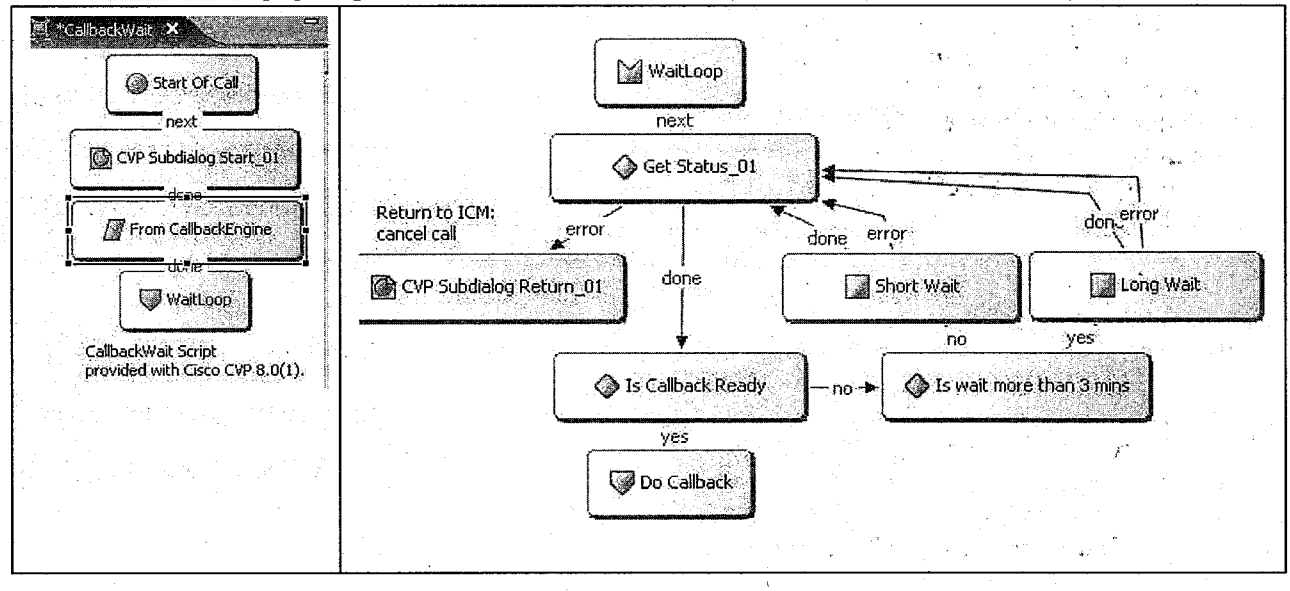
**The Studio Application CallbackEngine** says Goodbye, disconnects the caller's leg of the call, and then does an Application Transfer to the **CallbackWait** application. Do **NOT** modify **CallbackEngine** other than to set the Default Audio Path.




## CallbackWait Application

**A.** The **CallbackWait-Start** page uses an App Modifier element to rename variables passed from the **CallbackEngine** application. It then connects to the **WaitLoop** page.

**B.** The **CallbackWait-WaitLoop** page uses a custom element **Get Status_01** to determine the current wait time in the queue. It then either performs a short delay, a long delay, or connects to the **DoCallback** page to perform the callback.



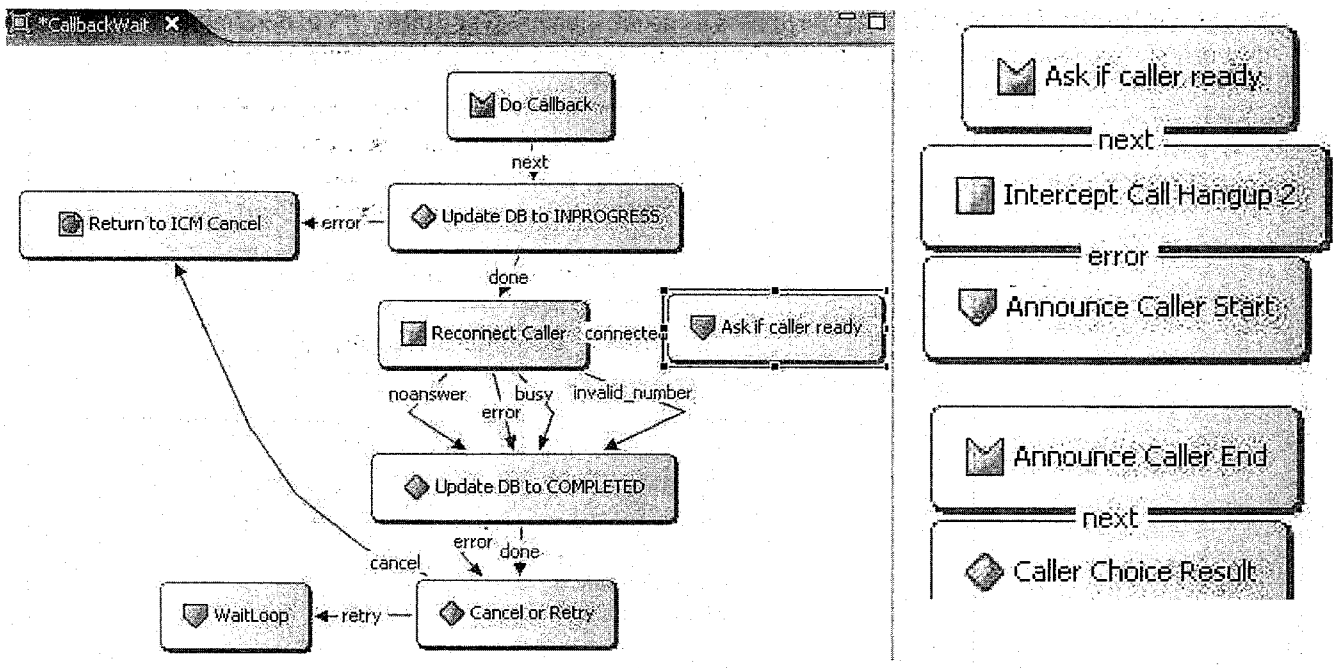
## C. CallbackWait Application – DoCallback Page

1. The **CallbackWait-DoCallback** page uses custom element **Update DB in PROGRESS** to update the database with status:INPROGRESS
2. **Reconnect Caller** is a custom element that instructs the system to perform the callback.
3. If someone answers the callback phone (**connected** exit state), then a page connector goes to the **Ask if caller ready** page.
4. All other **Reconnect Caller** exit states (noanswer, error, busy, invalid_number) connect to a custom element that updates the database with status:COMPLETED and the reason. This element creates element data **result** with the value **cancel** if the number of callback tries are exceeded.
5. **CancelOrRetry** is a decision element that tests if the Element data **result** equals **cancel**, if so it returns **cancelcall** to ICM. Otherwise connects to the Wait Loop (see above).

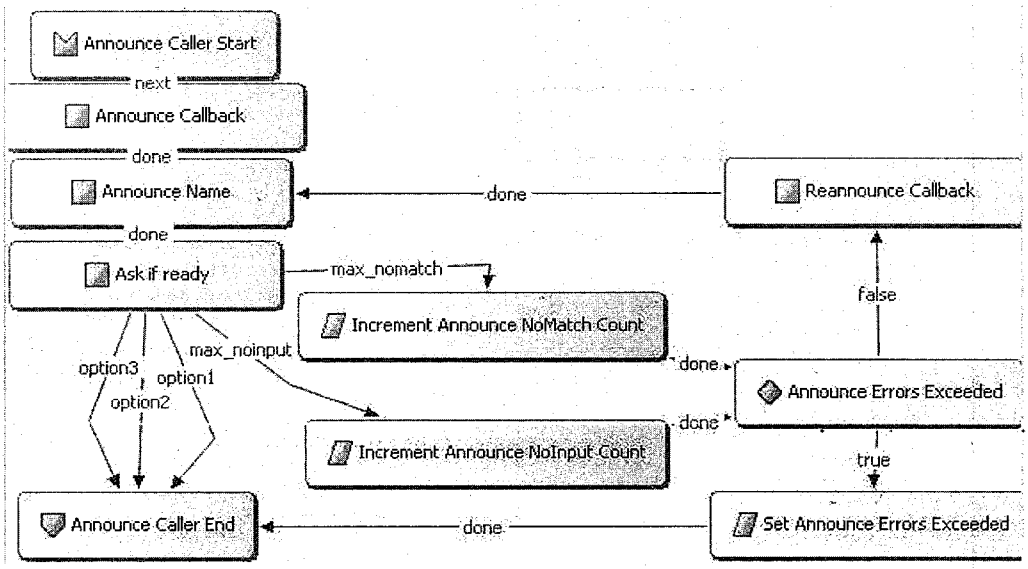
## D. CallbackWait Application – Ask if caller ready Page (Part 1)

The **Ask if caller ready** page uses a custom element to intercept if the caller hangs up and then connects to the **Announce Caller Start** (located on the **Announce Caller** page).

**DoCallback Page** **Ask if caller ready Page**

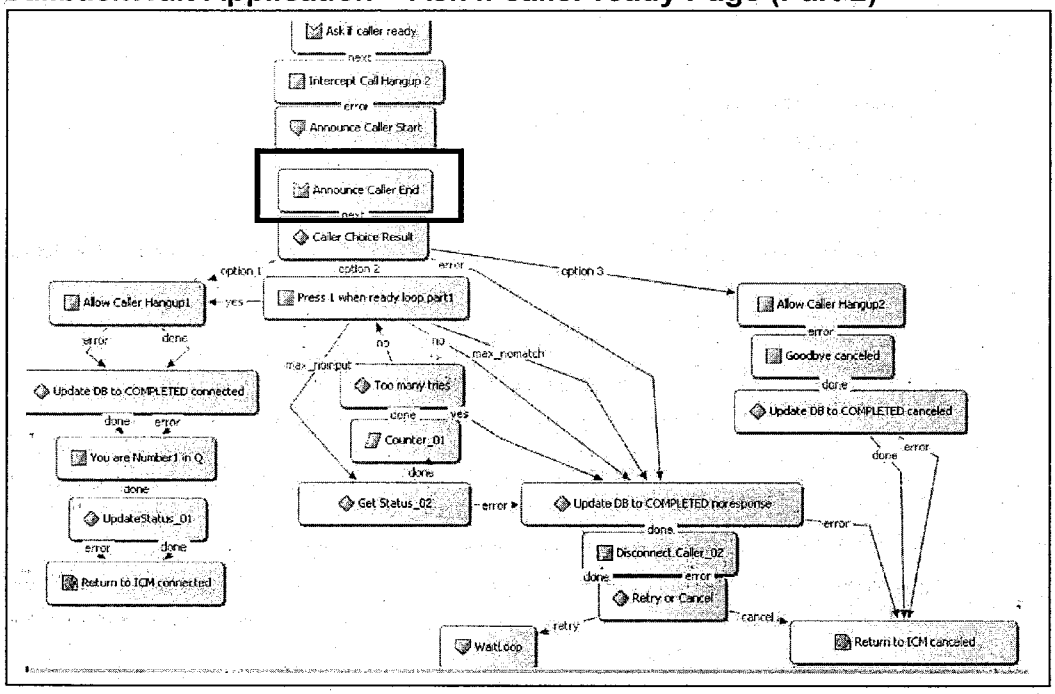


## E. CallbackWait Application – AnnounceCaller Page




1. **Announce Callback** is an audio element that plays CCAskIfCallerReadyPart1.wav (*This is a callback for*)
2. **Announce Name** is an audio element that plays the caller's recording (containing their name).
3. **Ask if ready** is a 3-option menu that plays CCAskIfCallerReadyPart2.wav (*Press 1 if you are ready to speak with an associate, press 2 if you need time to bring that person to the telephone, press 3 to cancel this callback*). The settings allow only 1 nomatch or 1 noinput.
4. **On nomatch or noinput**, this connects to a Math element to increment Session data named AnnounceNoMatchCount or AnnounceNoInputCount by 1.
5. **Announce Errors Exceeded** is a decision element that checks if BOTH counters are < 3. If so, it executes these elements again.
6. If caller press 1, 2, or 3 at the menu then this connects back to the **Announce Caller End** entry point on the **Ask if caller ready** page.

## D (ii). CallbackWait Application – Ask if caller ready Page (Part 2)




1. **Caller Choice Result** uses a decision element to test the caller's menu selection.
   - If the caller pressed 1: the **Update DB to COMPLETED** uses a custom element to change the DB to status:COMPLETED; reason:connected
2. **You are Number 1 in Q** plays the audio item CCYouAreNumber1InQueue.wav (*You are currently number 1 in the queue and an associate will be with you shortly*)
3. **Update Status 01** uses a custom element to change the DB to status:DROP FROM QUEUE
4. **Return to ICM connected** is a CVP Subdialog Return that returns **Caller Input: true**, and callback type, queuapp, qname, qtime in the FromExtVXML array.
   - If the caller pressed 2 (needs time): then the **Press 1 when ready loop part 1** element executes a YesNo_Menu that plays the audio **CCPress1WhenReady.wav** (*Press 1 when you are ready to continue*) with a 5s Noinput timer and Max Noinput count of 5.
   - The **max_noinput** exit state connects to a custom element **Get Status_02**, and then increments **Counter_01**, and then uses a decision to test if the counter is >= 3. If not, it executes this loop again. NOTE - this gives the caller a total of 15 Noinputs or 75 seconds (5 noinputs * 5s Noinput timeout * 3 loops = 75 seconds) to press 1 to continue the call.
5. If the caller uses up all the Noinput timeouts or an error occurs, then **Update DB to COMPLETED noresponse** element uses a custom element to update the DB to status:COMPLETED with reason:noresponse. This element also creates element data **result** with the value **cancel** if no further callback attempts should be made.
6. **Disconnect Caller_02** is a custom element that instructs the system to disconnect the caller's leg of the call.

7. **Retry or cancel** is a decision element to determine whether the element data **result** (step 5) equals **cancel** or not:
   - If **result** does not equal **cancel** then connect to the **Wait Loop** page to try another callback later.
   - If **result** does equal **cancel** or if the caller selected to cancel the callback, then **Return to ICM cancelled** is a CVP Subdialog Return that returns the following variables **Caller Input:true FromExtVxml0:cancelcall**

## ICM Routing Script - Continued

7. ICM checks if the caller rejected the callback (FromExtVxml[0]='cancelcall'). If so, ICM Releases the Call.

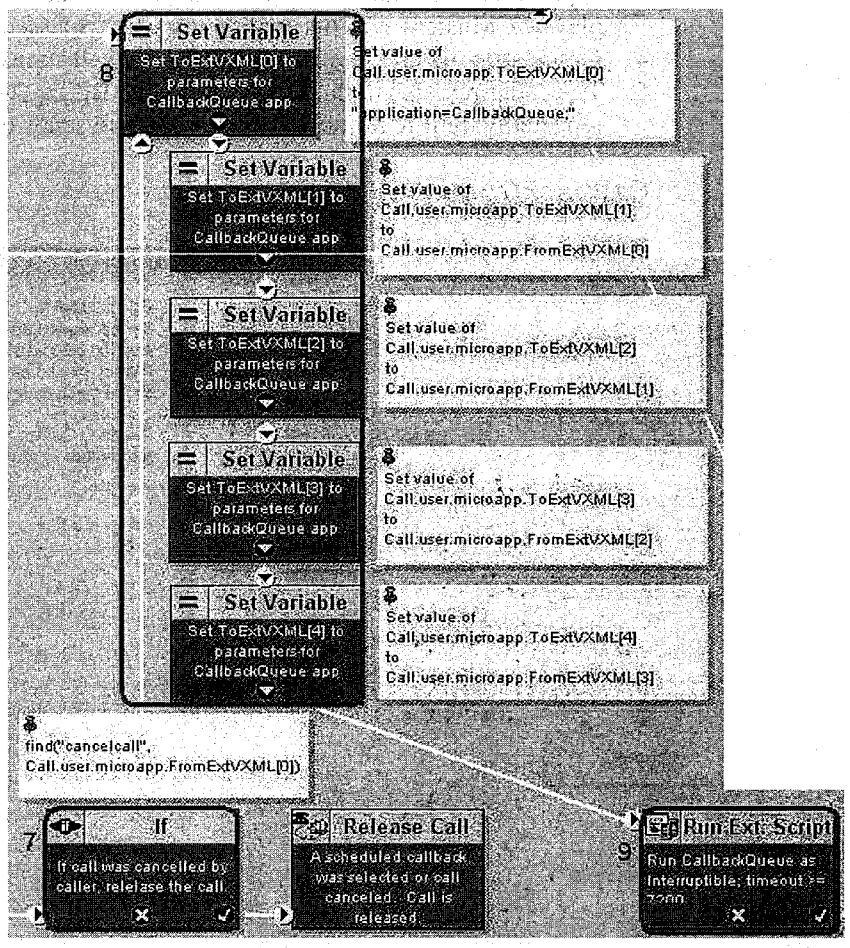
8. If the caller has accepted the callback, set variables to invoke the VXML Server **CallbackQueue** application.

9. **RunExtScript** invokes the **CallbackQueue**.

A) The **CallbackQueue** app contains an infinite loop – continually executing a **Subdialog Invoke** to the **queueapp** application assigned in step 3 above (**BillingQueue**), and then retrieving updated queue status for the call.

B) The **BillingQueue** app plays queue music to the caller until ICM interrupts and pulls the caller out and sends the call to an available agent.

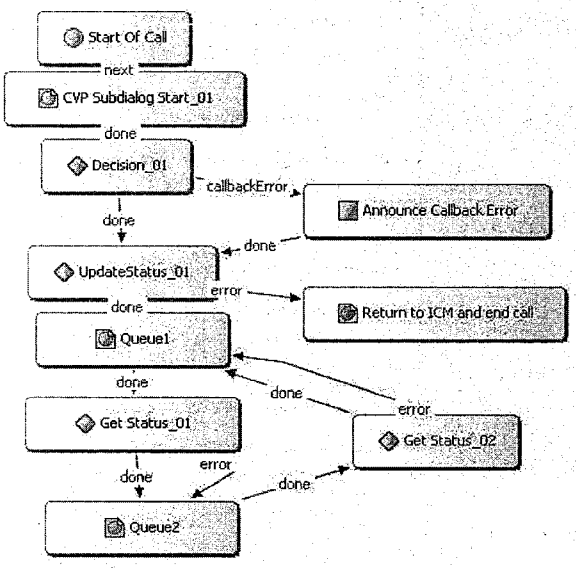
C) You may modify the **BillingQueue** application to play different music to the caller.



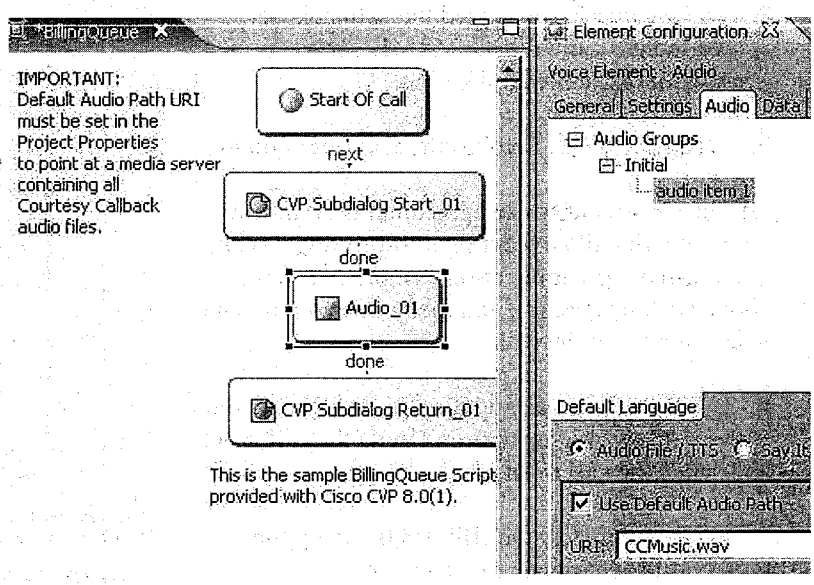
## CallbackQueue Application

1. **Decision_01** tests if it received ccError from ICM.. If so, it plays **CCError.wav** (*I'm sorry, the callback has been cancelled. If you'd like to wait for an associate, simply stay on the line*).
2. **UpdateStatus_01** updates the DB to status:ADD TO QUEUE
3. **Queue1** and **Queue2** are a Subdialog Invoke elements that invoke the **BillingQueue** application (or whatever application was configured in ICM in the parameter **queueapp**) that plays CCMusic.wav. This application was discussed as Application (2) above.
4. **Get Status_01** and **GetStatus_02** get the call status and continue.
5. The caller remains in this loop hearing the BillingQueue application's **CCMusic.wav** file repeatedly until ICM pulls the caller out and sends them to an agent.

**CallbackQueue (5)**

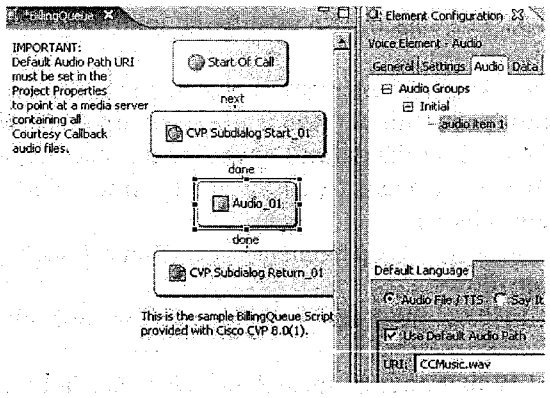



**BillingQueue (2)**



## BillingQueue Application

The **BillingQueue** application (invoked from the page above) plays audio configured in the **Audio_01** element and then returns back to the calling application (**CallbackEntry** – **Queue1** element). This application can be modified to play a different audio file for different queues.




8. Otherwise ICM invokes the **CallbackQueue** application which invokes the **BillingQueue** application repeatedly to play music to the caller (for a short time, hopefully) until the agent is available. Then ICM interrupts and sends the caller to the agent.

## CCB Studio Applications - Customization

### CallbackEntry Application

You may modify the initial IVR treatment a caller receives when they are presented with opportunity for a callback. You should **also** modify the following:

1. **Callback_Set_Queue_Defaults** Element - Located on the **Start of Call** page of the **CallbackEntry application**, this element is responsible for updating the DBServlet with the values that should be used for the **default** queue (or for other named queues, if you like). The defaults are used whenever a queue type is encountered for which there are no explicitly defined values. Settings include:
   A. **Maximum Percentage:** Max percentage of calls in callback queue that will be offered a callback. Default:100.
   B. **Maximum Count:** Maximum number of concurrent callbacks to allow. Default: 9,999,999.
   C. **Max Estimated Wait Time:** Estimated wait time threshold (in seconds) that will trigger the system to prompt the caller for a callback. Estimated times greater than this trigger a callback prompt.
   D. **Timezone:** Specify the time zone for the times given in the Sunday through Saturday time ranges (described in the following bullets).

CAUTION

   E. **Keepalive Interval:** Maximum keepalive interval in seconds.
   **CAUTION!!!** The value for **Keepalive Interval** must be greater than the length of the queue music that is played by the queuing application (BillingQueue in this example).

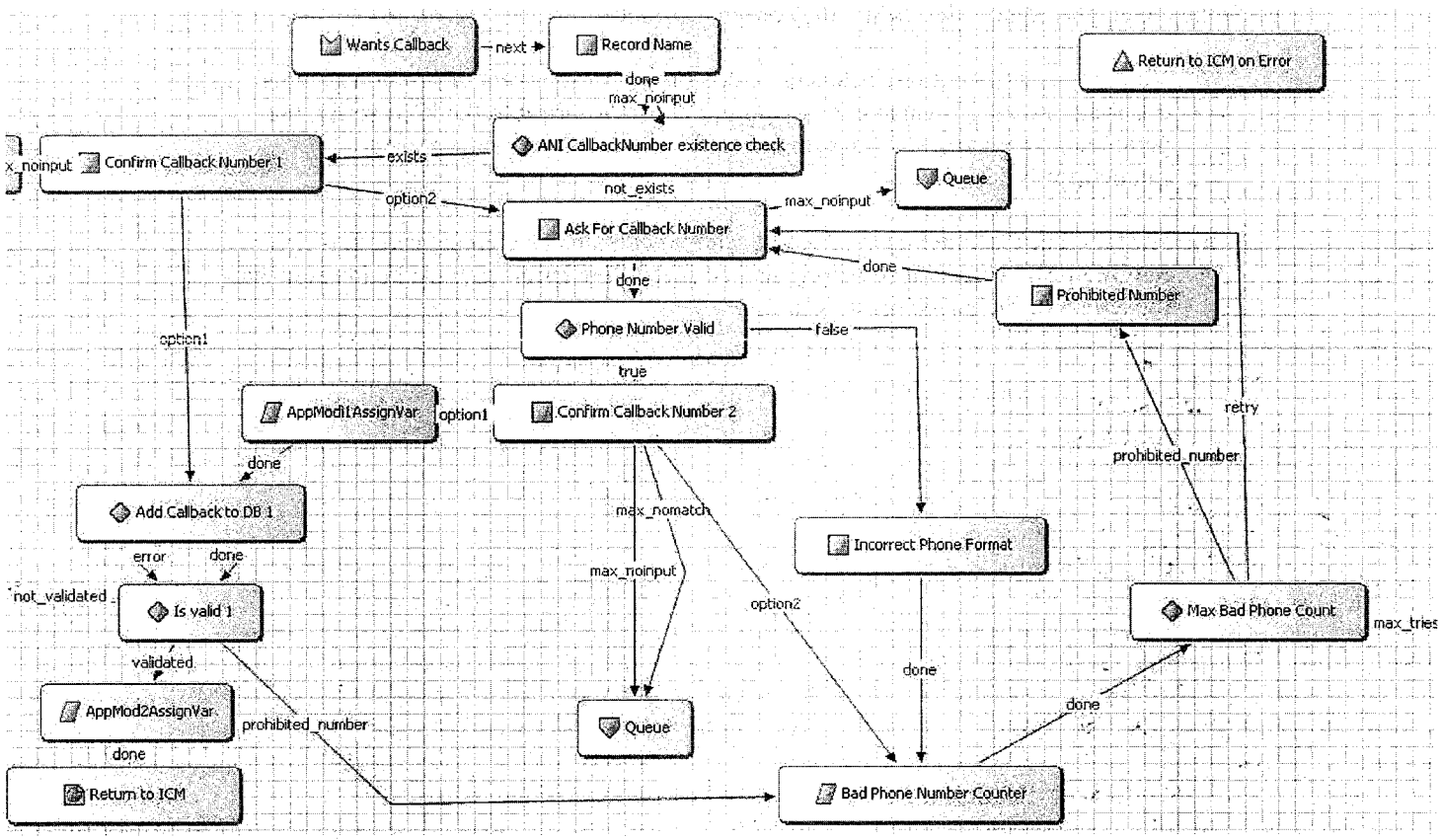
   F. **Reconnect Time:** Average time in seconds a callback takes, from the time of initial callback dialing to the time the caller is on the phone and has accepted the callback. This value is used in computing the optimum time for commencing the callback.
   G. **Calling Line ID:** Enter the CLI that should appear on the phone of the person being called back.
   **Note:** For the PSTN switch to display this number to the caller, it may be necessary for the service provider to specially provision it. Otherwise, typically, the default DID main number is used as the CLI.

   H. **Ring No Answer Timeout:** Enter the ring-no-answer timeout to be used for the callback. Default: 30 seconds.
   I. **Sunday (through Saturday) Time Range:** Specify the valid time range that callbacks maybe offered. Use 24 hour time notation. When the current time + the estimated wait time falls within this range, a callback is offered.
   J. **Max No Response Count, Max Busy Count, Max No Answer Count, Max Trunks Busy Count**, and **Max Error Count:** The maximum number of callback attempts and the interval (in seconds) between attempts when one of these types of failures is received on a callback.

   **Example, "3;60"** indicates **3** attempts at callback, **60** seconds apart. Callback attempts that fail due to **calling an invalid number** are the only type of failure that cannot be configured for multiple retries and will be limited to just the one failed callback.

2. **Preemptive Prompt1** and **Preemptive Prompt2** voice elements may be modified to announce the exp wait time and ask caller if they'd like a callback.

3. **CallbackEntry –Wants Callback** page. You may modify the caller interaction elements in addition to those below:




a) **Record Name** element settings (stores a recording of the caller's name).
   - **Path** - Enter the path to a web server or application server directory on the VXML Server where you want to store the recorded names of the callers. By default, the recording is saved into the folder **C:\Cisco\CVP\VXMLServer\Tomcat\webapps\CVP\audio**.

b) **Add Callback to DB 1** modify to reflect modifications to the Record Name element Settings:
   - **Recorded name file - Full or partial URL** pointing to the recording just created **Example**, /CVP/audio/{Data.Element.RecordName.filename} or http://IP/en-us/app/{Data.Element.RecordName.filename}
   - **Recorded name path** - Enter the file path (in Windows format) to the directory of the recorded file just created. The file name may be included, but is not necessary. **Example**, {Data.Element.RecordName.filepath} or C:\Inetpub\wwwroot\en-us\app
   - **Note - Recorded name path** should **only** be configured if you'd like auto-deletion of recordings in this folder based upon retention settings configured in the element

**CallbackWait Application**
Modify the IVR treatment a caller receives when they are called back. The caller interaction elements on the **AskIfCallerReady** page and **AnnounceCaller** page may be modified.

**BillingQueue Application**
Specify the music played to callers while they are in the queue.

a) You may create multiple instances of this project if you want different hold music for different clients. You also need to modify the ICM script. In the ICM script, the parameter **queueapp=BillingQueue** would also have a counterpart, eg **queueapp=SalesQueue**.

b) NOTE - The **CallbackEntry** Project (above) contains an element called **SetQueueDefaults** whose Setting named **Keepalive Interval** must be **greater** than the length of the queue music you use.

## (Reference) The Studio CCB Elements (Folder: Elements/Cisco/Callback/)

1. **Callback_Add** element adds a **callback object** to the database after all the callback information has been collected from the caller. In addition, it can be optionally configured to automatically delete old recorded files at specified intervals.

   A. Settings
   - **Callback Number** - caller's phone number for a callback
   - **Recorded Name File** – URL to recorded audio to play back to caller when calling back
   - **Recorded Name Path** - Leave this blank to disable auto-delete of recorded audio files.
     Otherwise enter a file path (not the URL). All files beginning with 'audio' in this directory are auto-purged based upon Retention settings:
   - **Recorded File Retention** – Number minutes to retain recorded file before it's eligible for deletion
   - **Recorded File Deletion Interval** – Number of interval minutes for checking whether to delete recorded files.

   B. Element Data: **result** –will contain one of these values: valid, no_validation and invalid_time
   - valid – callback object was added to the database
   - no_validation - occurs when a callback object cannot be created because Callback_Validate element was not executed in the script.
   - invalid_time –the time selected for the scheduled callback is invalid.

2. **Callback_Enter_Queue** element is responsible for adding a new caller to the queue. This element must be executed for all callers even if the caller may not be offered a callback.
   - Element Data: **ewt** – estimated wait time passed from ICM

3. **Callback_Validate** element is responsible for verifying whether or not a callback can be offered to the caller during this call. Depending on the outcome of the validation, the Validate element exits with one of four states.

   C. Exit States:
   - **preemptive** – the callback is valid and can be offered during this call
   - **refresh** - The validation could not be performed because the DBServlet needs a reference data refresh. The app must call **SetQueueDefaults** before validation can occur.
   - **none** – callback not allowed
   - **error** – a value couldn't be retrieved

   D. Element Data:
   - **result** – exit state result
   - **ewt** – estimated wait time passed from ICM
   - **gw** – gateway identifier
   - **loc** – gateway location info
   - **capacity** – gateway capacity

4. **Callback_Disconnect_Caller** element is responsible for disconnecting the caller's leg of the call. The IP leg of the call for CVP is preserved to hold the caller's place in line until the callback is made back to the caller
   A. Element Data**: result** – status of the request

5. **Callback_Get_Status** element is responsible for retrieving all information about the callback related to the current call (if a callback exists).

   a) startCallback -Specifies whether the application should call the caller, given current caller position in queue and rate of de-queue.
   b) ewt - Current estimated remaining wait time in seconds for this caller before the callback should be initiated.
   c) qpos- Current position in queue.
   d) rec - Recording URL that was stored in the callback table. This only needs to be returned if startCallback is true.
   e) DORateA- Average number of seconds for each caller in this queue to leave the queue. This includes both callers leaving queue by going to agents and callers in queue abandoning.
   f) DORateB - Average number of seconds for the #1 caller in this queue to leave the queue.
   g) RORate - Average number of seconds that it takes to get the caller back after starting the callback. The rate is the same for all queues. This includes dial time, ring time, and IVR time spent asking the caller if they are ready to take the callback.
   h) cli - The Calling Line ID to be used for this callback
   i) rna - Ring No Answer timeout for this call
   j) dn - Destination number for this outbound call

6. **Callback_Update_Status** element is responsible for updating the database after a callback disconnect or reconnect.
   B. Settings
      - **Status**: one of {PENDING, INPROGRESS, COMPLETED, ADD TO QUEUE, or DROP FROM QUEUE}
      - **Reason**: only used for status:COMPLETED. One of {error, busy, noanswer, noresponse, invalid_number, connected, trunkbusy, caller_cancelled}

   C. Element Data
      - **Result** - value will be one of {cancel, retry, done}

7. **Callback_Wait** element is responsible for sleeping the application for X seconds. The application hands control back to cvp_ccb_vxml.tcl with the parameter wait=X.

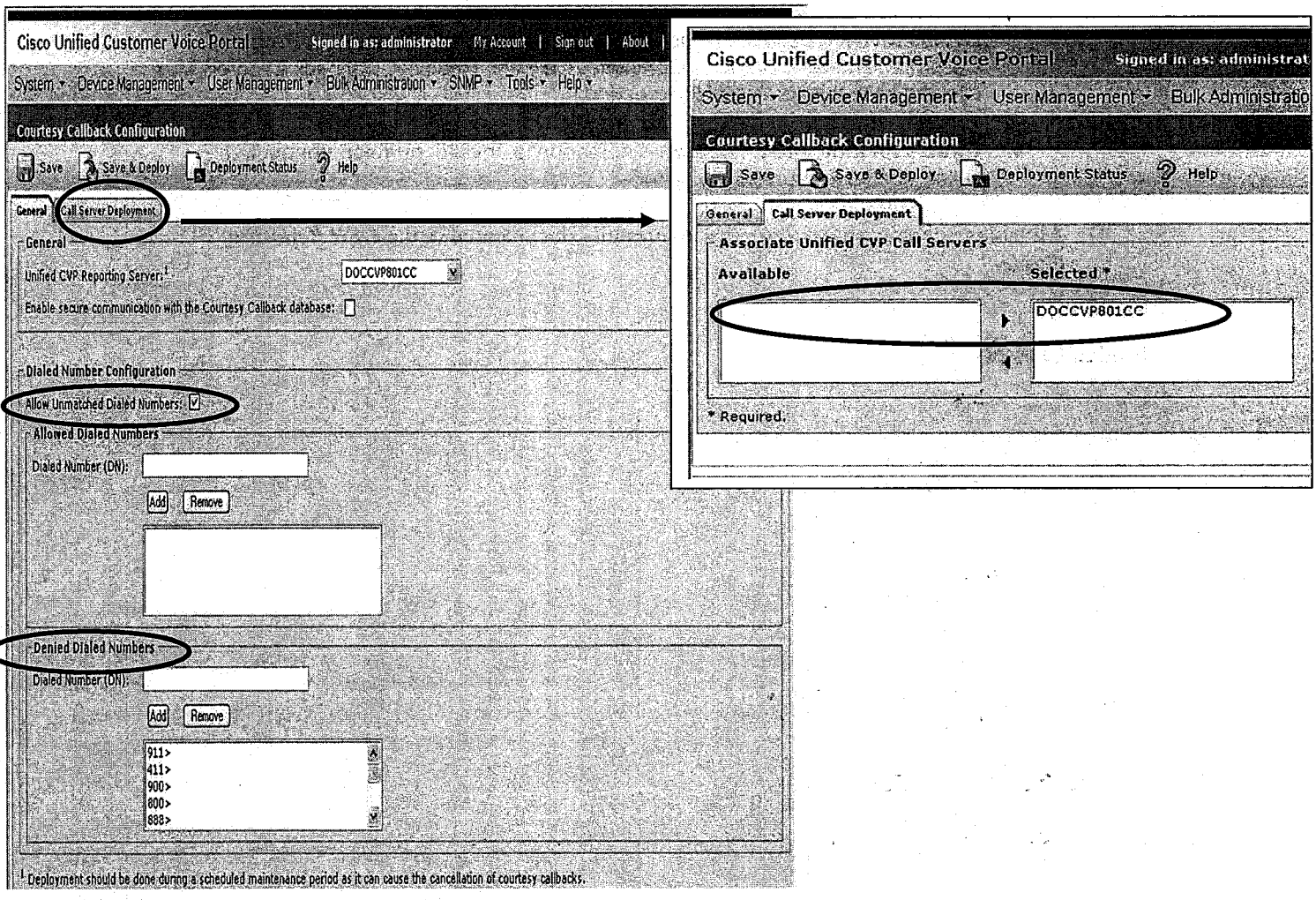
8. **Callback_Reconnect** element is responsible for reconnecting the caller's leg of the call.

For more details, see the ***CVP Studio and VXML Server Element Specifications*** manual.

## Reporting Server Configuration for Courtesy Callback

1. Use the Operations Console to configure the Reporting Server Database for Courtesy Callback. **https://10.1.78.10:9443/oamp**

2. Select **System > Courtesy Callback**
   A. **General**: Select a Reporting Server to use for the CCB database from the list.
   B. **General**: Optionally enable secure communication with the CCB database
   C. **Dialed Number Configuration**: Specify allowed and disallowed phone numbers for CCB Use > as a wild card. To allow all phone numbers except those disallowed, you MUST click the checkbox **Allow Unmatched Dialed Numbers.**
3. Select the **Call Server Deployment** tab and move the Call Server to use for Courtesy Callback from the **Available** column to the **Selected** column.
4. Press **Save** to deploy and activate the CCB database on the next restart of the Reporting Server. Or press **Save & Deploy** to activate it now. Note restarting a Reporting Server could cause the cancellation of currently scheduled callbacks.



## Reporting Server CCB Database

There are 3 tables in the **Courtesy Callback Database** of the **Reporting Server**.

A. The **Callback** table. When a caller requests a callback and records their name and enters the callback phone number, that request is stored in the **Callback** table. There is one row for each callback attempt made.

B. The **CallBackQueue** table holds data for the queue in which the call sits until a slot becomes available. This table also includes a column named ValidationStatus with the result of the Studio application's Get_Status query. These values include: ICM_NO_PREEMPTIVE_ALLOWED, NOT_IN_QUEUE, TOD (callback not allowed for the requested time of day), EWT (estimated wait time is too short to offer a callback), PROBE_FAILED_NO_RESPONSE, PROBE_FAILED_NO_CONFIG, EXCEED_CAPACITY_GW, EXCEED_CAPACITY_QUEUE,

C. The **CallBackEvent** table holds one record for each callback event that occurs for the call. For each event, there is an **Event** and a **Cause** logged:

- **Events include**:Callback Canceled, Callback Pending, Callback In Progress, Callback Tentative, Callback Complete, Callback Recover, Callback Created,Max allowed callbacks to this ANI exceeded.

- **Causes include**: busy, noanswer, noresponse, invalid_number, connected, caller_canceled, trunksbusy, error

D. All tables, schema, and sample queries are fully documented in the ***CVP Reporting Guide***.

[End of Chapter]

# Chapter 12

# Operations Console and Reporting Server Overview

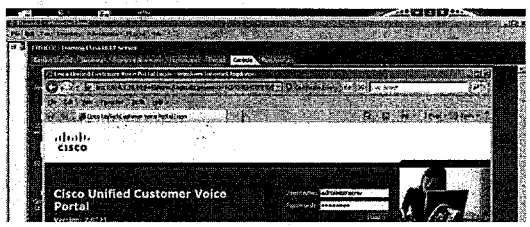
- Using the Operations Console to Deploy a VXMLServer Application
- Configuring the Reporting Server
- Configuring VXMLServer information to Send to the Reporting Server
- Working with Reporting Server Tables

## Using the Operations Console to Deploy a Studio App

1. You can use the to **CVP Operations Console** deploy one or more Studio applications to one or more VXML Servers, where it runs the deployApp.bat or updateApp.bat script immediately.

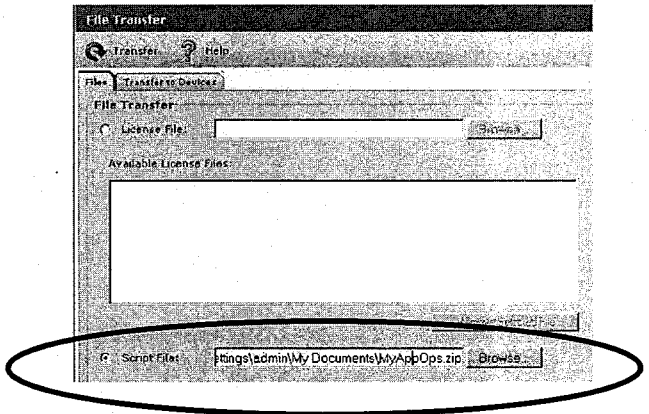
2. From **Studio**, right-click the application name in the Navigator and select **Deploy**.
   a) From the Deploy screen, select one or more applications to deploy.
   b) Select the radio button **Deploy Destination**: **Archive File**:
   c) Use the **Browse**… button to specify the directory: **Desktop**
   d) Enter the File Name: **MyApp**
   e) Press **Finish**
   f) This stores the selected applications into one zip file **MyApp.zip**.

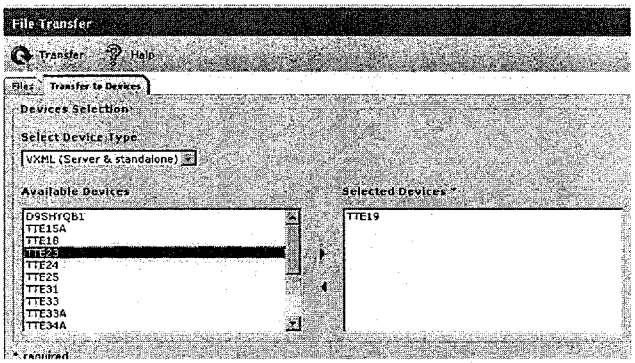
3. Launch a web browser and enter the URI for the classroom **Operations Console**:
   - https://10.1.78.10:9443/oamp
   - Username: **administrator**
   - Password: **TTE4you!**




4. In the Operations Console, select the menu **Bulk Administration / File Transfer**


   A. Select **Script File.** Browse to Desktop. Select **MyApp.zip**




   B. **Transfer to Devices**
      - Select Device Type: **Vxml Server**
      - Holding down the Ctrl key, select the Vxml Servers from the **Available Devices** pane

- Press the right arrow, to move them to the **Selected Devices** pane

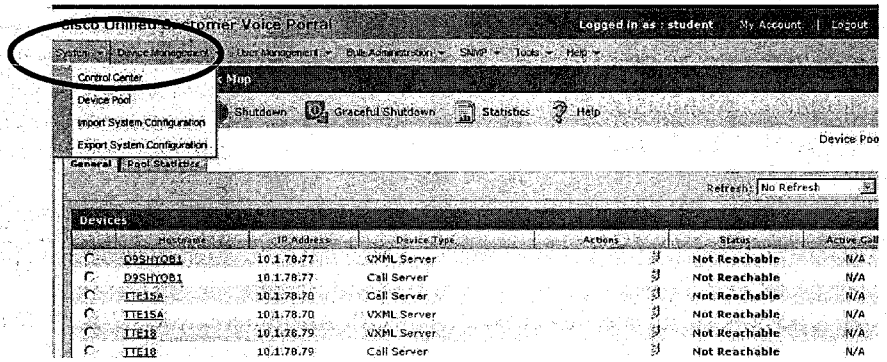
C. Press the **Transfer** button in the bottom right corner.

D. **Only ONE PERSON at a time can transfer files!!!**

E. The zip file is transferred to the range of VXML Servers, it is unzipped, and the appropriate Admin script (deployApp.bat or updateApp.bat) is automatically run.

5. The Ops Console can also be used for administration, such as Shutdown and Start of each configured component.

- Select the menu **System / Control Center** to view a list of configured components and their status, the number of current callers, and administrative functions.



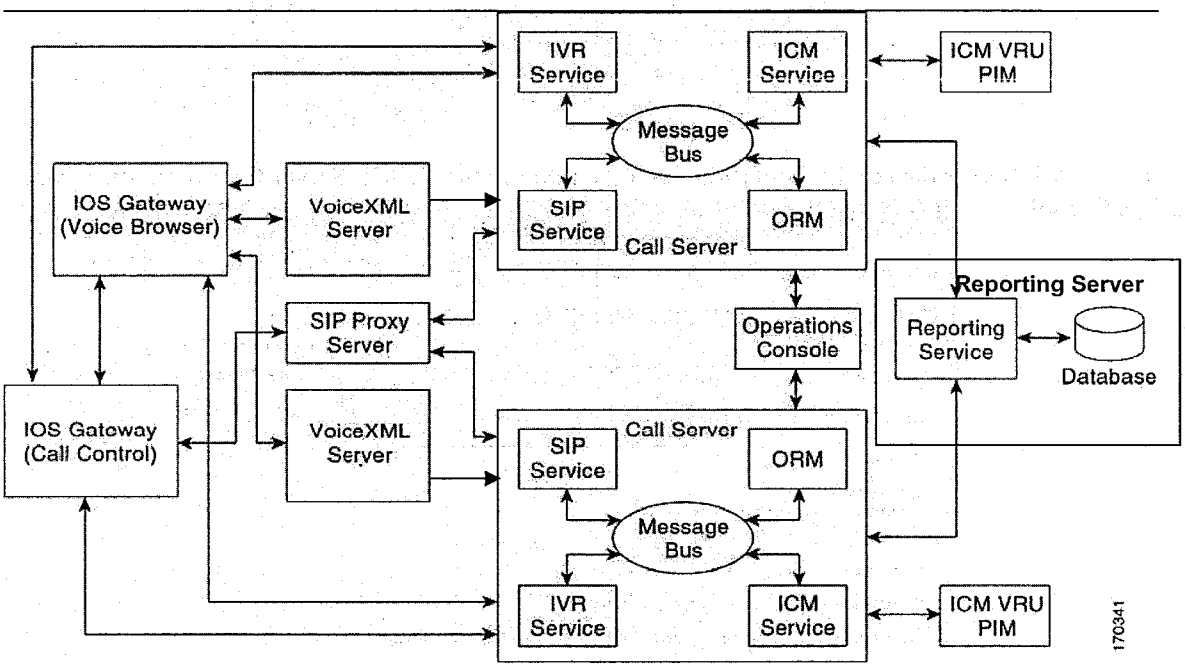
## The Reporting Server

1. The Reporting Server stores CVP VXML Server data for reporting purposes. It consists of the **Reporting Service**, and hosts an **IBM Informix Dynamic Server (IDS)** database management system for historical reporting.

2. A third-party reporting engine, such as Crystal Reports or **Cisco Unified Intelligence Center CUIC**, can be used to generate reports.

3. The database schema is fully published in the ***Cisco Reporting Guide for CVP*** reference manual.

4. The Reporting Service receives reporting data from the **IVR Service, the SIP Service** and the **VXML Server**.

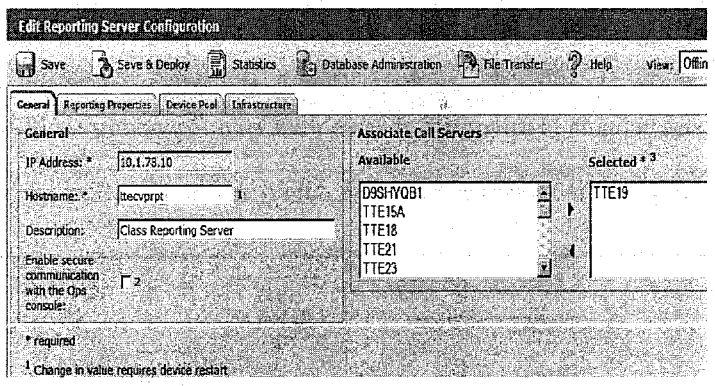



Note - The Ops Console is actually connected to each managed CVP component through an OAMP Resource Manager (ORM) that is co-located on each managed CVP component and is invisible to the user.

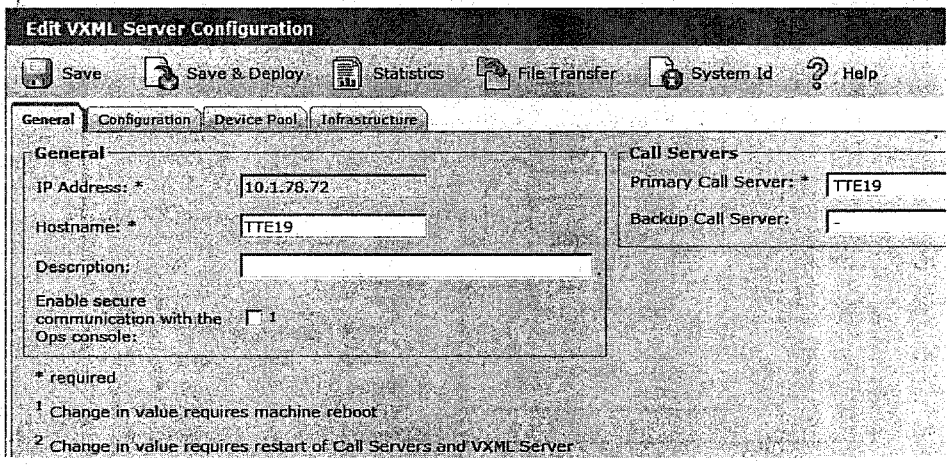
5. It is important to synchronize clocks across all CVP components using something like NTP.

6. You only need one Reporting Server. If more than one is used, be aware that:
   - Each Call Server and each VXML Server can be associated with only one Reporting Server
   - Reports cannot span multiple Informix databases
   - .During temporary database outages, messages are buffered on the Call Server and inserted into the database when the it comes back on-line.

7. Adding a Reporting Server to the Ops Console
   A) Choose **Device Management > CVP Reporting Server.**
   B) Each Reporting Server has one or more **Call Servers** associated to it.
   C) Select the **Reporting Properties** tab and configure reporting properties.
   D) Optionally, select the Infrastructure tab and configure log file and syslog settings.
   E) Click Save & Deploy

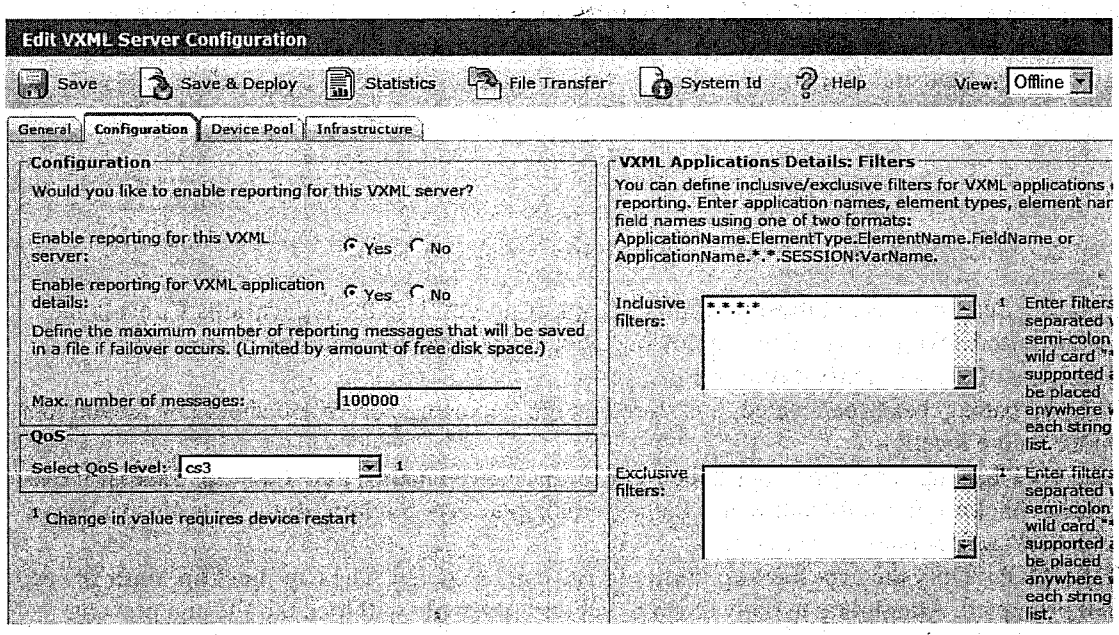



8. To save VXML Server data to the Reporting Server, configure the VXML Server filter properties from the Ops Console.




   A) On the Operations Console configure **filters** to control the data that the VXML Server feeds to the Reporting Server.
      i. Choose **Device Management > VXML Server.**
      ii. Choose the VXML Server that you want to edit.
      iii. Click **Edit.** The VXML Server Configuration window opens to the General Tab.
      iv. Select the **Configuration** tab. Then configure VXML Server properties.
      v. Select the **Yes** button for **Enable reporting for VXML application details** (default 'No')
      vi. In the **VXML Applications Details: Filters** pane, enter a semi-colon separated list of **inclusive filters** to define the data sent to the Reporting Server.
      vii. Optionally, enter semi-colon separated list of **exclusive filters** to exclude some of the data specified by the inclusive filter.
      viii. **Click Save & Deploy** to save and apply the changes to the VXML Server.

ix. **Restart** the VXML Server and the primary and backup Call Servers.



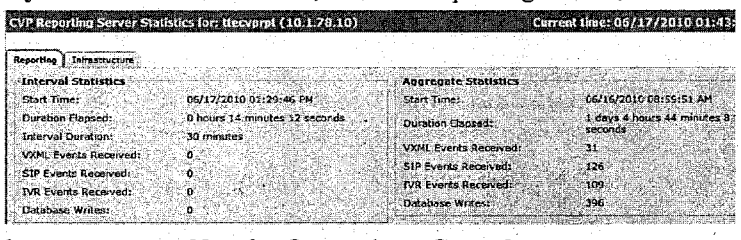

B) **Inclusive and exclusive filters** operate using the following rules:
   i. By default, **all items are excluded** from reporting data unless they are added to an Inclusive Filter. This is true for all items except the Start, End, CVP_Subdialog_Start/End elements.
   ii. The Exclusive Filter takes precedence over the Inclusive Filter.

C) The syntax for Inclusive/Exclusive filters is:
**Appname.ElementType.ElementName.FieldName**

   i. The **Element Types** are: Decision, Action, Custom, HotLink, HotEvent, ElementFlag, Voice, VXMLInsert, ReqICMLabel, General
   ii. You may include * as wildcard, and instead of ElementName to indicate Session Variables
   iii. A semicolon (;) should be used to separate multiple items in a filter and to separate multiple filters. For example, `AppName.*.*.SESSION:var1;var2`
   iv. A wildcard (*) can be specified anywhere within the application name, element type, element name, or field name. For example, `AppName.*.*.SESSION:g*`
   v. Syntax Examples:
      - `MyApp.Voice.*.*` All voice elements in MyApp
      - `MyApp.*.*.get*` All fields in MyApp starting with the string `get`
      - `MyApp.*.*.SESSION:var1` The session variable named var1

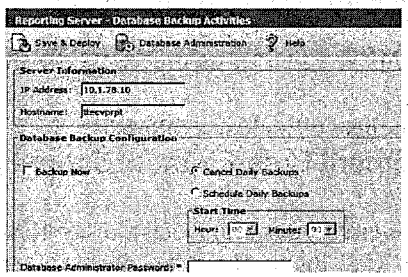
   vi. Element types, element names, and field names can contain alphanumeric characters, underscores, and a space character. **An application name can NOT contain the space character.**

9. **Reporting Server Statistics** include the total number of events received from the IVR, SIP, and VoiceXML services. To view the statistics select **System > Control Center**, select a Reporting Server, and then click the Statistics icon in the toolbar.




10. The Reporting Service does not itself perform backups or purges. Use the **Operations Console** to perform database backups, purge data, deploy licenses, and to manage users and passwords.
    A) **Backups of the Informix database:**
    - When a backup has been created, it is stored on the Reporting Server and should be moved to a more secure location.
    - The CVP backup script first copies the most recent backup named **cvp_backup_data** (if it exists) to **cvp_backup_data.old** and writes the new backup to **cvp_backup_data**. This always leaves two backup files on the local system.
    - CAUTION: Only the **cvp_backup_data.old** file can be copied. The **cvp_backup_data** file can *not* be copied. Attempting to copy the **cvp_backup_data** file will lock the file and prevent another backup from running.
    - Perform during low call volume times.
    - To restore the database you must run an Informix command from the Reporting Server, not from the Operations Console.

CAUTION: By default, scheduled backups are turned **off** !




    B) **Schedule Informix data purge** from the Ops Console.
    - Data purge is **required** and must be scheduled at least 3 hours in advance through the Ops Console.
    - CVP categorizes data into different levels. A high level category, such as Call, cannot have a lower retention time than a dependent category, such as CallEvent.. Default retention times, in days, is in the last column below.

**Reporting Server Tables (For a detailed schema, see the Cisco Reporting Guide)**

1. The Reporting Server installs with a menu and text driven interface named **dbaccess** for issuing SQL commands to access to the contents of the database. This can be useful to test if data is being stored into the Reporting Server.

2. Log into the Reporting Server Database using dbaccess:
   - Start > Run > dbaccess
   - Connection > Connect > (select) cvp_db_tte
   - User Name > **cvp_dbadmin**
   - Password> **pwd**
   - Select a Database > **cvp_data@cvp_db_tte**
   - Exit
   - Query Language > New > **select * from vxmlElementDetail** (pr
   - Run

3. Below are some of the important tables with important fields highlighted.

**Call (GUID – Date/Time/Ani/Dnis)**

| | |
|---|---|
| **callguid** | **8909C53D100001217A8339E70A014E3C** |
| callstartdate | 02/28/2005 |
| startdatetime | 2009-05-28 17:04:27.515 |
| enddatetime | 2009-05-28 17:05:07.031 |
| calltype | A |
| **ani** | **4072** |
| **dnis** | **2072** |
| numoptout | 0 |
| numonhold | 0 |
| numtimeout | 0 |
| numerror | 0 |
| numappvisited | 1 |
| totaltransfer | 1 |
| dbdatetime | 2005-02-28 17:04:27.000 |

**********

**CallEvent (GUID – Event – Cause)**

| | |
|---|---|
| **callguid** | **8909C53D100001217A8339E70A014E3C** |
| callstartdate | 02/28/2005 |
| callLegid | ZDMzNTk4NmU4NGYxMjRiNTZjMThjNWJjZmI1Mz |
| eventdatetime | 2009-05-28 17:04:27.515 |
| subsystemtypeid | 0 |
| **eventtypeid** | **0 (new call)** |
| **causeid** | **0** |
| dbdatetime | 2005-02-28 17:04:27.000 |
| mediafilename | NA |

**********

| | |
|---|---|
| **eventtypeid** | 4 (Answer) |

**EventTypeID, Name**

0, "New Call"
1, "Connect Failure"
2, "Busy"
3, "No Answer"
4, "Answer"
5, "Abandon"
6, "Disconnect"
7, "Hang Up"
8, "App Transfer"
9, "App Session Complete"
10, "Call Transfer"
11, "Run Script"
12, "Agent Recording"
13, "ICM Recording"

**CauseID, Name**

0, "None"
1, "Normal Completion"
2, "Call Abandon"
3, "Call Transferred"
4, "New Transaction"
5, "Busy"
6, "No Answer"
7, "Maintenance"
8, "Net Congestion"
9, "Net Not Obtainable"
10, "Reorder Tone"
11, "Resources Not Available"
12, "Trunks Busy"
13, "Called Party Disconnected"
14, "Max Ports"
15, "Suspended"
16, "Time Out"
17, "Invalidated"
18, "Error"
19, "Video Answered"
1001, "Hang Up"
1002, "Network"
1003, "System"

**causeid** **0**
**********

**eventtypeid** **9** **(App session complete)**
**causeid** **1**
**********

**VxmlElement (ElementID – ElementName-ElementType-Exit State )**

elementid 1001341243605227937
callstartdate 03/01/2005
callguid 8C6D486E100001213062FB7E0A014
sessionid 1001351243605227937
**elementname** **start**
enterdatetime 2009-05-29 09:53:47.937
exitdatetime 2009-05-29 09:53:47.937
**elementtypeid** **0**
**resultid** **1**
exitstate next
*******

**elementname** **CVP Subdialog Start_01**
elementtypeid 2
resultid 1
exitstate done
*******

**elementname** **CVP Subdialog Return_01**
elementtypeid 3
resultid 2
exitstate none
**********

**ElementTypeID, Name**

0 Start
1 End
2 Subdialog_Start
3 Subdialog_Return
4 Decision
5 Action
6 Custom
7 HotLink
8 HotEvent
9 ElementFlag
10 Voice
11 VXMLInsert
12 ReqICMLabel
13 General

**ResultID, Name**

1, "Normal"
2, "Invalidated"
3, "HotEvent"
4, "HotLink"
5, "Hang Up"
6, "Error"
7, "Transfer"

**VxmlElementDetail (ElementID – ElmtData varName,Value)**

elementid 1001071243612703343
**varname** **value** **(Element variable name)**
**varvalue** **1214?** **(Element variable value)**
vardatatypeid 0
actiontypeid 1
eventdatetime 2009-05-29 11:58:23.343
dbdatetime 2005-03-01 10:56:40.000
callstartdate 03/01/2005

**VxmlSession (Studio SessionID – GUID (ICM Callid))**

sessionid 1001101243605043921
callstartdate 03/01/2005
callguid 8C6A797F10000121378A543B0A014E3C
startdatetime 2009-05-29 09:50:43.921
enddatetime 2009-05-29 09:50:47.093
appname MyApp

| | |
|---|---|
| sessionname | 10.1.78.72.1243605043921.1.MyApp |
| sourceappname | |
| eventtypeid | 7 |
| causeid | 1 |
| duration | 3 |
| dbdatetime | 2005-03-01 08:48:57.000 |

**VxmlSessionVariable (ElementID – Session Variable Name, Value)**

| | |
|---|---|
| sessionid | 1001021244063676187 |
| elementid | 1001041244063676281 |
| **varname** | **totalPrice** |
| **varvalue** | **0.00** |
| vardatatypeid | 0 |
| eventdatetime | 2009-06-03 17:14:36.281 |
| **fromicm** | **f** |
| actiontypeid | 1 |
| dbdatetime | 2005-03-06 16:12:32.000 |
| callstartdate | 03/06/2005 |

**VxmlVoiceInteractDetail (ElementID – AudioGroup Spoken)**

| | |
|---|---|
| elementid | 1001041243612695718 |
| elaspedtimemillis+ | 8 |
| **voiceactiontypeid** | **3** |
| value | **initial_audio_group** |
| dbdatetime | 2005-03-01 10:56:28.000 |
| callstartdate | 03/01/2005 |

**********

**VoiceActionTypeID, Name**
1, "No Match"
2, "No Input"
3, "Audio Group"
4, "Input Mode"
5, "Utterance"
6, "Interpretation"
7, "Confidence"

**CustomContent (entries from the AddToLog field)**

| | |
|---|---|
| elementid | 1001091244063348203 |
| varname | ***LoggingString= |
| varvalue | ;rx=1111;rx=2222; |
| eventdatetime | 2009-06-03 17:09:19.312 |
| dbdatetime | 2005-03-06 16:07:16.000 |
| callstartdate | 03/06/2005 |

**VxmlError**

| | |
|---|---|
| elementid | 1001081243859880109 |
| errorname | javaApiError |
| description | An exception was thrown by convertToTTS of the Say It Smart plugin class AudumSayItSmarCtreditCard |
| eventdatetime | 2009-06-01 08:38:00.093 |
| dbdatetime | 2005-03-04 07:36:07.000 |
| callstartdate | 03/04/2005 |

[End of Chapter]

# Chapter 13

# Documenter Printout

### Documenter

1. You can use the **Call Studio Documenter** to print a rich text file of the application.

2. In Studio, **right-click** the project name in the **Navigator**.

3. Select **Documenter**. Then select the desired options to print:
   - Diagrams
   - Element configuration
   - Project properties, Hotlinks and Hotevents
   - Audio prompt listings
   - Revision History
   - Specify the ouput file to hold the RTF formatted file.

4. NOTE – while you can open this document in any text editor, you can only view the diagrams by viewing the output in Word.

# Sample Documenter Printout:

# Voice Application Specification

# Pharmacy Lab

**Mon, 17-Mar-2011**

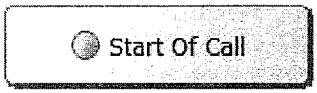
# Revision History

| Revision number | Change notes | Revised by | Date |
|---|---|---|---|
| | | | |

# Table of Contents

## 1. Call Flow Diagrams

### 1.1. Call Flow Diagram Legend

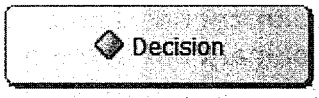



Start of Call - The single main entry point into the voice application.


Voice Element - A reusable, VoiceXML-producing dialog with a fixed or dynamically produced configuration.


Action Element - Encapsulates actions that execute on the application server


Web Service Element - An embedded web service interaction.




Decision Element - Makes decisions that are mapped to at least two exit states.


Flag - System records the traversal of the call through this element.


Hot Event - A global event that executes the developer-specified actions when it is caught.


Hot Link - A globally accessible utterance and/or key press that immediately brings the caller to a specified part of the call-flow or throws an event.


Application Transfer - Provides "virtual" transfers to other voice applications on the same VXML Server.


Element Group - Represents a consolidated group of elements.

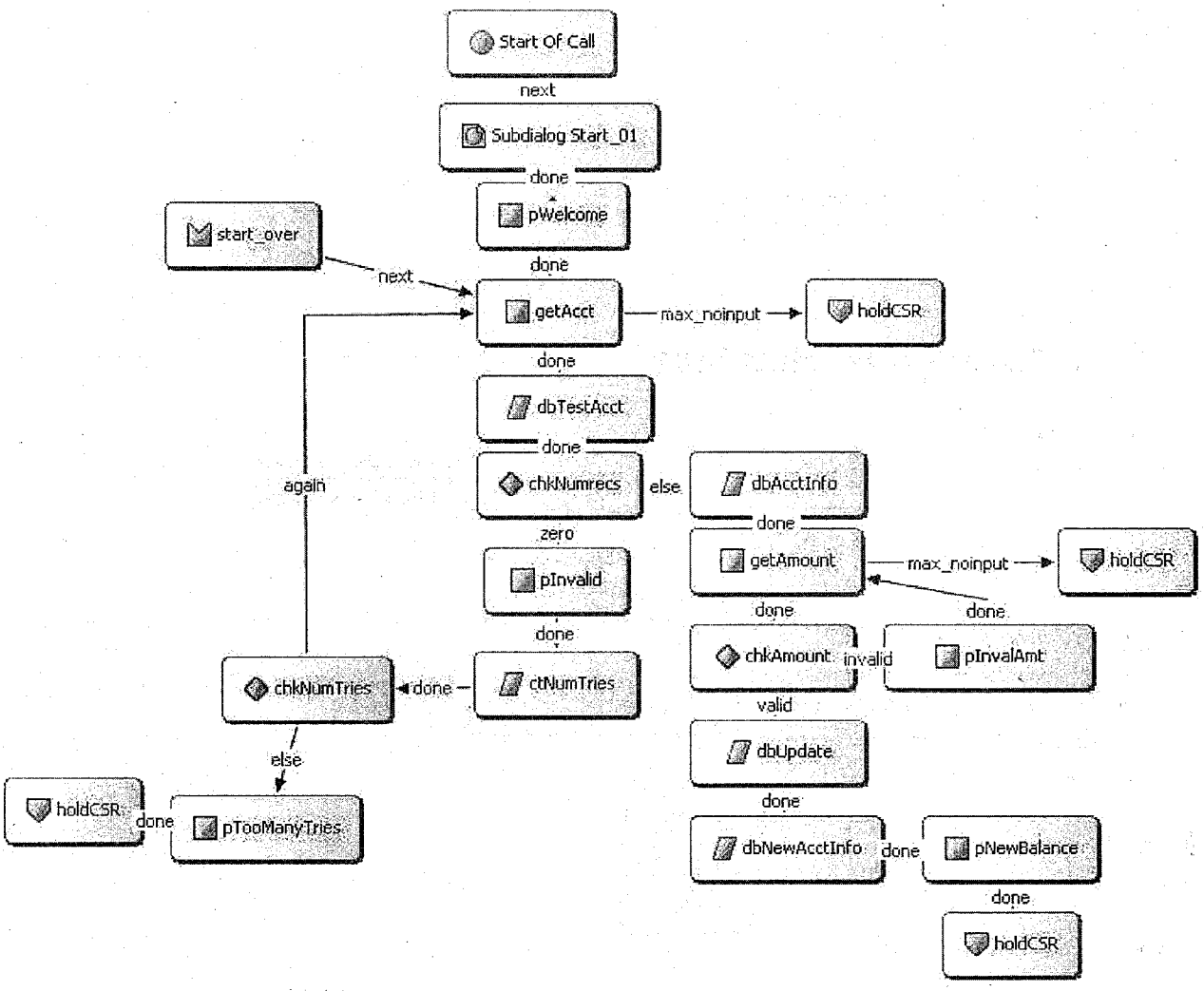

Page Entry - This represents an entry point into a particular page in the call flow workspace.


Page Connector - This element provides connectivity to other pages via the "Page Entry" element.

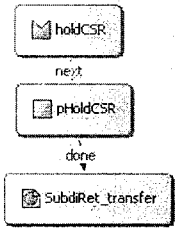

CVP SubdialogStart - This element is used as the entrance point to an application that is invoked as a subdialog.

# 1.2. Application Call Flow Diagrams

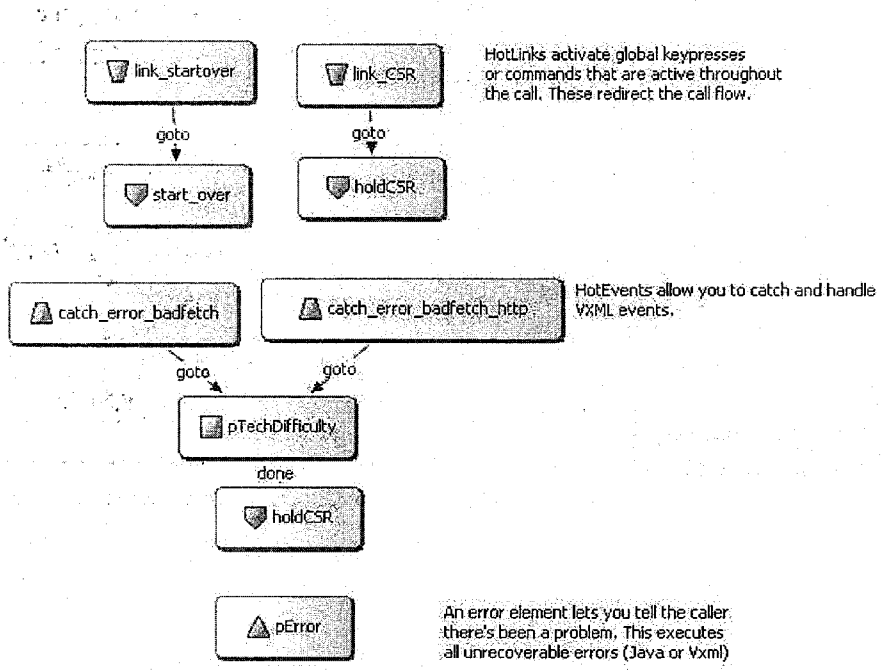
## 1.2.1. Page: page1



## 1.2.2. Page: holdCSR




## 1.2.3. Page: HotLinks



# 2. Global Configurations

## 2.1. Application Settings

### 2.1.1. General Settings

| Name | Value | |
|---|---|---|
| Deploy Version | CVP 7.0.2 | |
| Maintainer | | |
| Language | en-US | |
| Encoding | | |
| Subdialog | true | |
| Session Timeout | 30 minutes | |
| VoiceXML Gateway | Cisco Unified CVP 7.0 with OSR 3 | |
| User Management Enabled | no | |
| **Loggers** | | |
| ErrorLog | Class | com.audium.logger.application.error.ApplicationErrorLogger |
| | Enforce Call Event Order | no |
| | Include Configuration File | yes |
| | File Name | ErrorLogConfig.xml |
| | | |
| ActivityLog | Class | com.audium.logger.application.activity.ActivityLogger |
| | Enforce Call Event Order | yes |
| | Include Configuration File | yes |
| | File Name | ActivityLogConfig.xml |
| | | |
| | | |

### 2.1.2. Audio Settings

| Name | Value |
|---|---|
| Generic Error Message | Sorry. There has been an error. |
| Error Audio URI | /CVP/audio/error.wav |
| Suspended Message | Sorry, this voice application has been taken down for maintenance. |
| Suspended Audio URI | /CVP/audio/suspend_audio.wav |
| Initial On-Hold Audio URI | /CVP/audio/onhold_initial.wav |
| Main On-Hold Audio URI | /CVP/audio/onhold_continue.wav |
| Default Audio Path URI | |

### 2.1.4. Root Document Settings

| **JavaScript** | |
|---|---|
| | |
| **VoiceXML Property** | |
| Interdigittimeout | 5s |
| **VoiceXML Variable** | |

# 3. Element Configurations

## 3.1.1. Page: page1

### getAcct - Voice Element

| | | |
|---|---|---|
| **General** | | |
| Dynamic Configuration | no | |
| Set UID | no | |
| **Settings** | | |
| Input Mode | both | |
| Noinput Timeout | 5s | |
| Digits Max NoInput Count | 3 | |
| Digits Max NoMatch Count | 3 | |
| Digits Confidence Level | 0 | |
| Min Digits | 4 | |
| Max Digits | 4 | |
| Disable Hotlinks | false | |
| Secure Logging | false | |
| Maxnbest | 1 | |
| **VoiceXML Property** | | |
| interdigittimeout | 5s | |
| **Prompts** | | |
| Digits Initial (Default) - audio item 1 | Audio File/TTS | |
| | Use Default Audio Path | yes |
| | URI | |
| | TTS | enter your 4 digit account number. |
| | Comments | |
| | Language | Default |
| Done (Default) - you entered | Audio File/TTS | |
| | Use Default Audio Path | yes |
| | URI | |
| | TTS | you entered |
| | Comments | |
| | Language | Default |
| Done (Default) - getAcct.value as Digits | Say It Smart | |
| | Data | {Data.Element.getAcct.value} |
| | Type | digits |
| | Input Format | number |
| | Output Format | digits |

| | | |
|---|---|---|
| | Use Recorded Audio | no |
| | Use Default Audio Path | yes |
| | Audio Path | |
| | Audio Fileset | standard |
| | Audio Type | wav |
| | Comments | |
| | Language | Default |
| **Data** | | |
| | | |
| **Local Hotlinks** | | |
| | | |
| **Source Exit State** | | |
| again | chkNumTries | |
| done | pWelcome | |
| goto | link_startover | |
| **Target Exit State** | | |
| done | dbTestAcct | |
| max_nomatch | pHoldCSR | |
| max_noinput | pHoldCSR | |
| **Comment** | | |
| Collect the caller's 4 digit prescription number. | | |

## chkNumTries - Decision

| **General** | |
|---|---|
| Decision Editor | <knowledge_base><br><rule name="chkNumTries" default_exit_state="else"><br><exit_state name="again" conjugate="and"><br><number operator="less"><br><data><br><element name="ctNumTries" variable="count" /><br></data><br><constant_number value="3" /><br></number><br></exit_state><br></rule><br></knowledge_base> |
| **Source Exit State** | |
| done | ctNumTries |

# 4. Recording List

## 4.1. Prompts

| Element | Audio Group (Language) | File Name | Transcript | Language | Comments |
|---|---|---|---|---|---|
| getAcct | Digits Initial (Default) | | enter your 4 digit account number. | Default | |
| getAcct | Done (Default) | | you entered | Default | |
| getAcct | Done (Default) | | <say it smart: Digit-By-Digit> | Default | |
| getPaymentAmount | Currency Initial (Default) | | the balance of your | Default | |
| getPaymentAmount | Currency Initial (Default) | | {Data.Element.dbAcctInfo.account1} | Default | |
| getPaymentAmount | Currency Initial (Default) | | account is | Default | |
| getPaymentAmount | Currency Initial (Default) | | <say it smart: Currency ($)> | Default | |
| getPaymentAmount | Currency Initial (Default) | | how much to withdraw | Default | |
| pInvalAmt | Initial (Default) | | invalid amount entered | Default | |
| pInvalid | Initial (Default) | | invalid account number entered | Default | |
| pNewBalance | Initial (Default) | | the balance of your {Data.Element.dbAcctInfo.account1} is now ${Data.Element.dbNewAcctInfo.balance1} | Default | |

## 4.1.2. Page: holdCSR

| Element | Audio Group (Language) | File | Transcript | Language | Comments |
|---|---|---|---|---|---|
| pHoldCSR | Initial (Default) | | hold for a C S R | Default | |

## 4.1.3. Page: HotLinks

| Element | Audio Group (Language) | File | Transcript | Language | Comments |
|---|---|---|---|---|---|
| pError | Initial (Default) | | | Default | |
| pTechDifficulty | Initial (Default) | | | Default | |

[End of Chapter]

{ } = ~~Squiggly Brackets~~

= Braces

= Curly Braces

( = Bracket

Round Bracket

DASH

HIPEN

~ TILDA / WAVE CINE

' APOSTROFIC / DOUBLE QUOTE