Avaya Experience Portal Programmer's Reference

Release 8.1.2.3
Issue 1
August 2025

Notice

While reasonable efforts have been made to ensure that the information in this document is complete and accurate at the time of printing, Avaya assumes no liability for any errors. Avaya reserves the right to make changes and corrections to the information in this document without the obligation to notify any person or organization of such changes.

Documentation disclaimer

“Documentation” means information published in varying media which may include product information, subscription or service descriptions, operating instructions and performance specifications that are generally made available to users of products. Documentation does not include marketing materials. Avaya shall not be responsible for any modifications, additions, or deletions to the original published version of Documentation unless such modifications, additions, or deletions were performed by or on the express behalf of Avaya. End user agrees to indemnify and hold harmless Avaya, Avaya's agents, servants and employees against all claims, lawsuits, demands and judgments arising out of, or in connection with, subsequent modifications, additions or deletions to this documentation, to the extent made by End user.

Link disclaimer

Avaya is not responsible for the contents or reliability of any linked websites referenced within this site or Documentation provided by Avaya. Avaya is not responsible for the accuracy of any information, statement or content provided on these sites and does not necessarily endorse the products, services, or information described or offered within them. Avaya does not guarantee that these links will work all the time and has no control over the availability of the linked pages.

Warranty

Avaya provides a limited warranty on Avaya hardware and software. Please refer to your agreement with Avaya to establish the terms of the limited warranty. In addition, Avaya’s standard warranty language as well as information regarding support for this product while under warranty is available to Avaya customers and other parties through the Avaya Support website: https://support.avaya.com/helpcenter/getGenericDetails?detailId=C20091120112456651010 under the link “Warranty & Product Lifecycle” or such successor site as designated by Avaya. Please note that if the product(s) was purchased from an authorized Avaya channel partner outside of the United States and Canada, the warranty is provided by said Avaya Channel Partner and not by Avaya.

“**Hosted Service**” means an Avaya hosted service subscription that You acquire from either Avaya or an authorized Avaya Channel Partner (as applicable) and which is described further in Hosted SAS or other service description documentation regarding the applicable hosted service. If You purchase a Hosted Service subscription, the foregoing limited warranty may not apply but You may be entitled to support services in connection with the Hosted Service as described further in your service description documents for the applicable Hosted Service. Contact Avaya or Avaya Channel Partner (as applicable) for more information.

Hosted Service

THE FOLLOWING APPLIES ONLY IF YOU PURCHASE AN AVAYA HOSTED SERVICE SUBSCRIPTION FROM AVAYA OR AN AVAYA CHANNEL PARTNER (AS APPLICABLE), THE TERMS OF USE FOR HOSTED SERVICES ARE AVAILABLE ON THE AVAYA WEBSITE, HTTPS://SUPPORT.AVAYA.COM/LICENSEINFO UNDER THE LINK “Avaya Terms of Use for Hosted Services” OR SUCH SUCCESSOR SITE AS DESIGNATED BY AVAYA, AND ARE APPLICABLE TO ANYONE WHO ACCESSES OR USES THE HOSTED SERVICE. BY ACCESSING OR USING THE HOSTED SERVICE, OR AUTHORIZING OTHERS TO DO SO, YOU, ON BEHALF OF YOURSELF AND THE ENTITY FOR WHOM YOU ARE DOING SO (HEREINAFTER REFERRED TO INTERCHANGEABLY AS “YOU” AND “END USER”), AGREE TO THE TERMS OF USE. IF YOU ARE ACCEPTING THE TERMS OF USE ON BEHALF A COMPANY OR OTHER LEGAL ENTITY, YOU REPRESENT THAT YOU HAVE THE AUTHORITY TO BIND SUCH ENTITY TO THESE TERMS OF USE. IF YOU DO NOT HAVE SUCH AUTHORITY, OR IF YOU DO NOT WISH TO ACCEPT THESE TERMS OF USE, YOU MUST NOT ACCESS OR USE THE HOSTED SERVICE OR AUTHORIZE ANYONE TO ACCESS OR USE THE HOSTED SERVICE.

Licenses

The Global Software License Terms (“Software License Terms”) are available on the following website https://www.avaya.com/en/legal-license-terms/ or any successor site as designated by Avaya. These Software License Terms are applicable to anyone who installs, downloads, and/or uses Software and/or Documentation. By installing, downloading or using the Software, or authorizing others to do so, the end user agrees that the Software License Terms create a binding contract between them and Avaya. In case the end user is accepting these Software License Terms on behalf of a company or other legal entity, the end user represents that it has the authority to bind such entity to these Software License Terms.

Copyright

Except where expressly stated otherwise, no use should be made of materials on this site, the Documentation, Software, Hosted Service, or hardware provided by Avaya. All content on this site, the documentation, Hosted Service, and the product provided by Avaya including the selection, arrangement and design of the content is owned either by Avaya or its licensors and is protected by copyright and other intellectual property laws including the sui generis rights relating to the protection of databases. You may not modify, copy, reproduce, republish, upload, post, transmit or distribute in any way any content, in whole or in part, including any code and software unless expressly authorized by Avaya. Unauthorized reproduction, transmission, dissemination, storage, or use without the express written consent of Avaya can be a criminal, as well as a civil offense under the applicable law.

Virtualization

The following applies if the product is deployed on a virtual machine. Each product has its own ordering code and license types. Unless otherwise stated, each Instance of a product must be separately licensed and ordered. For example, if the end user customer or Avaya Channel Partner would like to install two Instances of the same type of products, then two products of that type must be ordered.

Third Party Components

The following applies only if the H.264 (AVC) codec is distributed with the product. THIS PRODUCT IS LICENSED UNDER THE AVC PATENT PORTFOLIO LICENSE FOR THE PERSONAL USE OF A CONSUMER OR OTHER USES IN WHICH IT DOES NOT RECEIVE REMUNERATION TO (i) ENCODE VIDEO IN COMPLIANCE WITH THE AVC STANDARD (“AVC VIDEO”) AND/OR (ii) DECODE AVC VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSED TO PROVIDE AVC VIDEO. NO LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE. ADDITIONAL INFORMATION MAY BE OBTAINED FROM MPEG LA, L.L.C. SEE HTTP://WWW.MPEGLA.COM.

Service Provider

WITH RESPECT TO CODECS, IF THE AVAYA CHANNEL PARTNER IS HOSTING ANY PRODUCTS THAT USE OR EMBED THE H.264 CODEC OR H.265 CODEC, THE AVAYA CHANNEL PARTNER ACKNOWLEDGES AND AGREES THE AVAYA CHANNEL PARTNER IS RESPONSIBLE FOR ANY AND ALL RELATED FEES AND/OR ROYALTIES. THE H.264 (AVC) CODEC IS LICENSED UNDER THE AVC PATENT PORTFOLIO LICENSE FOR THE PERSONAL USE OF A CONSUMER OR OTHER USES IN WHICH IT DOES NOT RECEIVE REMUNERATION TO: (i) ENCODE VIDEO IN COMPLIANCE WITH THE AVC STANDARD (“AVC VIDEO”) AND/OR (ii) DECODE AVC VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSED TO PROVIDE AVC VIDEO. NO LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE. ADDITIONAL INFORMATION FOR H.264 (AVC) AND H.265 (HEVC) CODECS MAY BE OBTAINED FROM MPEG LA, L.L.C. SEE HTTP://WWW.MPEGLA.COM.

Compliance with Laws

You acknowledge and agree that it is Your responsibility to comply with any applicable laws and regulations, including, but not limited to laws and regulations related to call recording, data privacy, intellectual property, trade secret, fraud, and music performance rights, in the country or territory where the Avaya product is used.

Preventing Toll Fraud

“Toll Fraud” is the unauthorized use of your telecommunications system by an unauthorized party (for example, a person who is not a corporate employee, agent, subcontractor, or is not working on your company's behalf). Be aware that there can be a risk of Toll Fraud associated with your system and that, if Toll Fraud occurs, it can result in substantial additional charges for your telecommunications services.

Avaya Toll Fraud intervention

If You suspect that You are being victimized by Toll Fraud and You need technical assistance or support, please contact your Avaya Sales Representative.

Security Vulnerabilities

Information about Avaya’s security support policies can be found in the Security Policies and Support section of https://support.avaya.com/security.

Suspected Avaya product security vulnerabilities are handled per the Avaya Product Security Support Flow (https://support.avaya.com/css/P8/documents/100161515).

Trademarks

The trademarks, logos and service marks (“Marks”) displayed in this site, the Documentation, Hosted Service(s), and product(s) provided by Avaya are the registered or unregistered Marks of Avaya, its affiliates, its licensors, its suppliers, or other third parties. Users are not permitted to use such Marks without prior written consent from Avaya or such third party which may own the Mark. Nothing contained in this site, the Documentation, Hosted Service(s) and product(s) should be construed as granting, by implication, estoppel, or otherwise, any license or right in and to the Marks without the express written permission of Avaya or the applicable third party.

Avaya is a registered trademark of Avaya LLC.

All non-Avaya trademarks are the property of their respective owners.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Downloading Documentation

For the most current versions of Documentation, see the Avaya Support website: https://support.avaya.com, or such successor site as designated by Avaya.

Contact Avaya Support

See the Avaya Support website: https://support.avaya.com for Product or Cloud Service notices and articles, or to report a problem with your Avaya Product or Cloud Service. For a list of support telephone numbers and contact addresses, go to the Avaya Support website: https://support.avaya.com (or such successor site as designated by Avaya), scroll to the bottom of the page, and select Contact Avaya Support.

Contents

Chapter 1: Introduction

Purpose

This document provides information about designing speech applications for Avaya Experience Portal. This document includes information about speech applications in Avaya Experience Portal, call classification, SIP application support, and Web Services.

This document is intended for application developers who want to design and implement speech applications. The document provides guidelines for designing speech applications and also includes information for call classification support, SIP application support and application web services provided by Avaya Experience Portal.

Chapter 2: Designing speech applications to run in Avaya Experience Portal

Speech applications in Avaya Experience Portal

Speech applications are the “directors” of Avaya Experience Portal system operations. When a caller dials in to the system, the Media Processing Platform (MPP) accesses the appropriate speech application to control the call. From that point on, the speech application directs the flow of the call until the caller hangs up or the application is finished.

Experience Portal systems can have more than one application active and available at a time. The MPP that takes the call accesses the appropriate application based on the Dialed Number Identification Service (DNIS).

Note:

An application does not have to have a DNIS assigned to it. In this case, such an application handles any call that comes in to the system by means of a DNIS that is not assigned to any other application on the system. However, you can only have one such application on the system. If you attempt to configure a second application without a DNIS, the system generates an error.

In addition, if the speech application requires Automatic Speech Recognition (ASR) or Text-to-Speech (TTS) resources, the MPP contacts the appropriate ASR or TTS server through an Media Resource Control Protocol (MRCP) proxy server.

Call flow example

This call flow example shows how the Experience Portal system interacts with other systems to handle an automated telephone transaction.

1. A caller from the Public Switched Telephone Network (PSTN) dials a telephone number.
2. The PSTN routes the call to the Private Branch Exchange (PBX) associated with that number.

3. Using Voice over IP (VoIP), the PBX breaks the voice data into packets and sends them over the LAN to a Media Processing Platform (MPP) server in the Experience Portal system.
4. The MPP server looks at the Dialed Number Identification Service (DNIS) for the incoming call and uses the configuration information downloaded from the EPM server to match the number to a speech application that has been added to Experience Portal.
5. The MPP starts an Avaya Voice Browser session and passes it the Universal Resource Indicator (URI) specified for the selected speech application.
6. The Avaya Voice Browser contacts the application server and passes it the URI.
7. The application server returns a VoiceXML page to the Avaya Voice Browser.
8. Based on instructions on the VoiceXML page, the MPP uses prerecorded audio files, Text-to-Speech (TTS), or both to play a prompt to start interaction with the caller. For TTS, the MPP establishes a connection to a TTS server and the ASCII text in the speech application is forwarded for processing. The TTS server renders the text as audio output in the form of synthesized speech which the MPP then plays for the caller.

   Note:

   This connection requires one TTS license, which can be released as soon as processing is complete.
9. If the caller responds by:
   - Entering Dual-tone multi-frequency (DTMF) digits, the results can be processed locally by the MPP or passed to the ASR server for remote processing. The administrator selects local or remote DTMF processing when the application is configured. The digits entered are then returned to the application for further action.
   - Speaking, the MPP establishes a connection to an Automatic Speech Recognition (ASR) server and sends the caller's recorded voice response to the ASR server for processing. The ASR server then returns the results to the application for further action.

     Note:

     This connection requires one ASR license, which is *not* released until the entire call is complete.
10. If errors are encountered during the call, how these errors are handled depend on the type of grammar used by the application. If the application grammar is:
    - Dynamic, or In-line, the speech server gets the grammar directly from Experience Portal and any error messages are passed back to Experience Portal.
    - External or static, the speech server asks for the grammar using the URL specified in the application. If the URL points to an application server, the speech server interacts directly with that application server. Because Experience Portal is not involved in this communication, any error messages passed back by the application server may not be passed back to Experience Portal.
11. The application terminates the call when it finishes execution or when the caller hangs up.
12. When the call ends, the PSTN clears the call from the PBX and releases the ASR license if one was required.

Speech application development tools

Any speech application that is compliant with the VoiceXML Version 2.1 Recommendation or Call Control eXtensible Markup Language (CCXML) will run in an Experience Portal system, regardless of the tool in which the application was created. Avaya recommends that you create your speech applications with Orchestration Designer.

Orchestration Designer is an Eclipse plug-in that provides an integrated GUI for application design and implementation. It creates speech applications that automatically conform to the Experience Portal requirements and recommendations.

In addition, Experience Portal automatically includes all Orchestration Designer applications in the Application Summary report and Application Detail report. If you want these reports to display messages and status information from an application developed in a third-party tool, you must manually log the messages and status information from that application using the Application Logging web service.

Deploying a speech application

About this task

Before you can add a speech application to Experience Portal, you must deploy the speech application to an application server connected to your Experience Portal.

Procedure

1. Create the speech application.

   For design guidelines, see Design for user experience on page 14.

2. Package the application for deployment.

   For more information about packaging applications for deployment on Apache Tomcat or IBM WebSphere, see Tomcat and WebSphere speech application deployment guidelines on page 11.

3. Copy the speech application package file to the application server from which the application will run.

   Security alert:

   Observe appropriate security measures when copying application package files to the application server.

Next steps

Some application server environments require that you take additional steps to deploy the speech application after you copy the application files to the application server. The additional steps might include installing any run-time support files that the application requires. For more information about support files required by your application server, see the documentation for your server.

Tomcat and WebSphere speech application deployment guidelines

You can deploy Orchestration Designer speech applications on a Tomcat or WebSphere application server. If you want to use a different application server, consult your server documentation for deployment requirements.

Supported application servers

- Tomcat versions 7.0, 8.0, 8.5, and 9.0
- Boss Application Server 7
- Websphere 8.5.5 application server - with Java 7 support
- Websphere 8.5 application server - with Java 6 suuport by default, but has Java 7 option – see the Websphere documentation for details
- Websphere 9 application server - with Java 8 support
- Oracle Weblogic 10.3.6, 11g, and 12c

Apache Tomcat deployment guidelines

To deploy a speech application to an Apache Tomcat application server, you must package the application within a standard Web Archive (WAR) file. A WAR file is a compressed set of files, similar to a ZIP file. The WAR file format is specified by the J2EE specification and all J2EE-compliant application servers should support this format. The Tomcat servlet engine is optimized to handle WAR files. For more information about WAR file requirements for speech applications on your Apache Tomcat system, see the documentation for your system.

When you transport the WAR files to your Apache Tomcat application server, copy them to the directory *TomcatHome*\webapps\, where *TomcatHome* is the directory in which your Apache Tomcat application server software is installed.

Then, when you next start Tomcat, Tomcat automatically installs and deploys the application.

Important:

If you are redeploying an existing application, make sure the original application is not running before you deploy the new version on the server. If the original application is running, there could be conflicts with the log files.

IBM WebSphere deployment guidelines

In order to deploy a speech application on an IBM WebSphere or WebSphere Express application server, you must package the application within a standard Enterprise ARchive (EAR) file or Web ARchive (WAR) file with the JDK source level set to 15. These files are compressed sets of files, similar to a ZIP file. IBM WebSphere servlet engines can use either EAR or WAR file formats. For more information about EAR or WAR file requirements for speech applications on your IBM WebSphere system, see the documentation for your system.

When you transport the EAR or WAR files to your IBM WebSphere application server, make note of the directory to which you copy them. Then later, use the WebSphere Administrative Console to actually deploy the application from this location. For more information, see your IBM WebSphere documentation.

Important:

If you are redeploying an existing application, make sure the original application is not running before you deploy the new version on the server. If the original application is running, there could be conflicts with the log files.

Chapter 3: Speech application design guidelines

Best practices for speech application design

Sound files

High quality stereo sound files may appear to be the perfect way to communicate with your customers, but you must remember that these files will be played over a telephone line which has only 8 KHz bandwidth. Most of the higher frequencies are lost when a recording is played over the telephone.

When you record sound files:

- Record monaural rather than stereo.
- Use 16 bit-depth A-to-D conversion.
- Use a quality audio editing program to normalize the amplitude of your various phrases and convert to mu-LAW or A-LAW PCM.

When you have finished recording, listen to each recording in its final format. If you use the above guidelines, the way they sound at this point will be very close to how they sound over the telephone.

Supported wav file formats

Audio Format	Media Type
Raw (headerless) 8kHz 8-bit mono mu-law [PCM] single channel (G.711)	audio/basic
Raw (headerless) 8kHz 8 bit mono A-law [PCM] single channel (G.711)	audio/x-alaw-basic
WAV (RIFF header) 8kHz 8-bit mono mu-law [PCM] single channel	audio/x-wav
WAV (RIFF header) 8kHz 8-bit mono A-law [PCM] single channel	audio/x-wav
WAV (RIFF header) 8kHz 16-bit mono linear PCM single channel	audio/x-wav

Design for user experience

The best first step in planning an application is to envision exactly what you want the caller to experience when calling in to your system. Do not consider how to set up the application. Simply ask and try to answer as many questions as you can.

For example, ask and try to answer the following questions:

- What options do you want to offer callers?
- Do you want to offer callers the opportunity to interact in more than one language?
- What means do you want to offer callers to respond to options? By voice? By using touchtone keys on the telephone? By recording their answers or other short messages?
- What voice gender do you want to use in presenting your prompts?
- Do you need to use Text-to-Speech (TTS) technology to provide the caller a way to hear text-based information?
- What about hearing- or speech-impaired callers? How do you want to provide for their special needs?

Again, the idea is to ask as many questions as you possibly can. Create sample scenarios for the various situations you think callers might require help with. Try to be as comprehensive as possible.

Design for potential problems

One of the most important steps in planning a good speech application is to plan for any potential problem and error condition you can think of and to include error handlers that can deal with these issues. For example, how should the system respond when one of these problem situations arises?

- Technical or hardware limitations. What if the caller does not have a touchtone telephone? How does your system respond to TTY or TDD requests?
- Accessibility for callers with physical limitations. Have you allowed for the extra time it can take for callers with physical handicaps or other limitations?
- Language limitations. Is it likely that a caller will need to interact using a different language than the primary language? Do you have the necessary Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) servers and software to accommodate them?
- Personal preferences. Have you allowed for the personal preferences of your callers? Some people would rather interact with the system verbally, while others can prefer to use touchtone, or Dual-tone multi-frequency (DTMF) , responses. Other people prefer to interact with a live attendant no matter how good your Interactive Voice Response (IVR) speech application is. How easy is it for such users to get to a live attendant?

If an application encounters an error that is not handled by its own error handlers, or if an application cannot be started due to problems with the application server or the speech servers, Experience Portal uses the error handlers installed on the MPP server. In addition to designing

the speech applications, you should also design the error handlers that Experience Portal uses in these cases.

For example, you want your default event handler to:

1. Play a prompt explaining that there was a problem and that the customer is being redirected to an agent immediately.
2. Transfer the call to a special number reserved for such issues.

A call coming in on this special number alerts the agent that the caller has encountered and error in Experience Portal, and that the agent should find out what the customer was doing when the error occurred. The call center can then track these exceptions and fix areas that encounter frequent problems.

For more information, see Experience Portal event handlers on page 15.

Experience Portal event handlers

When a CCXML speech application tries to access an HTML page that cannot be found or a VoiceXML application encounters an unexpected event, the application responds with an exception error message. A well-designed speech application includes exception handlers that deal with these messages and help the application recover so that it can continue processing the call.

Experience Portal uses the error handlers defined in the application whenever possible. However, if an exception error message occurs that is not handled by the application, or if there is a problem running the application due to issues with the application server or the speech servers, Experience Portal uses one of the event handlers installed on the MPP server.

The event handler Experience Portal uses depends on the state of the speech application. If an application:

- Was successfully started and there is a call in progress, Experience Portal uses the event handler associated with that application when it was added to the Experience Portal system.
- Could not be started, Experience Portal looks at the type of application that was requested and uses the appropriate default CCXML or VoiceXML event handler.

When you install the software, Experience Portal automatically installs default event handlers for CCXML and VoiceXML, as well as an event handler prompt that is played by the default event handlers.

To customize the way Experience Portal reacts to a problem, you can add your own event handlers and prompts, and then designate which ones Experience Portal should use as the default. Customers can review and update the messages to match their voice and to say something more user-friendly than *The system is experiencing technical difficulty*.

For example, you want your default event handler to:

1. Play a prompt explaining that there was a problem and that the customer is being redirected to an agent immediately.
2. Transfer the call to a special number reserved for such issues.

A call coming in on this special number alerts the agent that the caller has encountered and error in Experience Portal, and that the agent should find out what the customer was doing when the error occurred. The call center can then track these exceptions and fix areas that encounter frequent problems.

Design for application flow

You have envisioned the experience you want your callers to have. You have tried to foresee and plan for any problem contingency that might arise. Now you are ready to start actually mapping the flow of your speech application. A number of methods can serve you well in this effort, but here are a couple of the more common approaches:

- Describe the flow verbally. Talk through each of your scenarios verbally. Make sure you take note of where the prompts occur and what you want callers to say or do. Record these verbal “walkthroughs”.
- Use a flow diagram. As you work through your scenarios, you can create a flow diagram to show the major points in the call flow. Use this diagram to show such things as:
  - Where you want to offer options to callers
  - Where you want them to listen to the entire prompt and where they can interrupt, or "barge in", and cut the prompt off
  - Where you require a response from callers and what the valid responses will be
  - Where you want or need to access databases to retrieve or record customer data
  - How and where you want to access a Web service to respond to a customer request

Again, the idea is to be as complete and comprehensive as possible. Try and foresee every eventuality, and map how the system will respond.

Design for modularity

When you are planning your speech application, be alert to places where you can reuse parts of the application in two or more places. Then, when designing and building the application, you can create these parts as modules that you can reuse wherever you need that functionality.

If you plan for these modules ahead of time, you can also develop them before developing your main application project file. That way, they are already available when you create your main application.

For example, you might want to collect bank account or credit card numbers from callers at several points in the call flow. You have figured out that it is the same basic process each time you need to collect such numbers. Therefore, you might want to create a speech project module that you can reuse in your master application whenever you need to collect this type of information from callers.

Using this modular approach to application design has several advantages:

- You can "develop once, use many times." This can be a tremendous advantage, especially if you have certain actions or options you want to offer in several places to your callers.
- It is easier to maintain the overall application, even if you are not reusing much of the code. When you use a modular approach, you can change one part of the application without necessarily having to rebuild the entire application.
- It can make it easier to debug your applications, by making it possible to isolate the trouble spots where errors are occurring.

A team of developers can work on separate pieces of an application separately and then merge their efforts.

Design for application resources

As a final step in planning your speech application, you must attempt to identify and list all the application resources you will need to develop the application. Application resources include components such as Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) servers, prerecorded phrases that the system plays back as prompts, and so on.

If you have planned the flow completely, keeping in mind all the resources you will need, you can list those application resources before you actually start to develop the application. Then you can create or import those resources before creating the call flow.

In the case of phrases, for example, if you know exactly what phrases you want to use, and if you plan to have those phrases recorded by a professional talent, you can list all the phrases and have them recorded as WAV files before you start to develop the actual application. Then, when you get to the points in the application where you need the phrases, you can import the prerecorded files.

CCXML and VoiceXML considerations

Keeping CCXML application sessions active until the MPP grace period expires

About this task

When you stop, restart, or halt an MPP, any CCXML application currently running on that MPP is sent a `ccxml.kill` event. This event notifies the application that the MPP is shutting down when the grace period expires.

If you do not define an error handler for this event in your CCXML application, the application exits immediately, even if it still processing a call.

Procedure

To keep the CCXML application active until the grace period has expired or until the application encounters an `</exit>` tag, add the following event handler to your CCXML application:

```
<transition event="ccxml.kill">
<!-- Do nothing.  Platform will kill session
after grace period-->
</transition>
```

Restrictions for dialogs attached to CCXML conference calls

Dialogs attached to conference calls may only play audio or text-to-speech prompts. If an attached dialog attempts to perform speech recognition processing, each attempt will result in a `no resource` error.

If you need speech recognition capabilities, you must attach the dialog to one of the calls that is part of the conference. In this case, however, the dialog can only process the speech that comes from the call to which it is attached.

Speech recognition results and VoiceXML applications

VoiceXML provides shadow variables accessible within a page to access recognition results such as `application.lastresult$`. While this includes a mechanism for presenting multiple possible semantic matches (n-best results whose presence can be determined by inspecting `application.lastresult$.length`), unfortunately the specification does not describe how to present a result that contains multiple interpretations for a given semantic match.

Avaya Experience Portal addresses the issue of a result containing multiple interpretations for a given semantic match by exposing the shadow variable `application.lastresult$.interpretation$`. To test for the presence of multiple interpretations in a result, examine `application.lastresult$.interpretation$.length`.

If the VoiceXML page containing that grammar uses the `maxnbest` property of `2`, then the recognition results arising from a caller saying “star gazer” would be accessed as follows in the VoiceXML specification:

```
application.lastResult$.length              2
application.lastResult$[0].confidence       0.83
application.lastResult$[0].utterance        Stargazer
application.lastResult$[0].interpretation   eyes
application.lastResult$[1].confidence       0.12
application.lastResult$[1].utterance        starchaser
application.lastResult$[1].interpretation   legs
```

Even though the recognition results contain the additional interpretations “look” and “run”, the VoiceXML specification makes no provisions for making those additional interpretations available to the application. To accomplish this, Avaya Experience Portal extends this list of shadow variables as:

```
application.lastResult$[0].interpretation$.length   2
application.lastResult$[0].interpretation$[0]       eyes
application.lastResult$[0].interpretation$[1]       look
application.lastResult$[1].interpretation$.length   2
```

```
application.lastResult$[1].interpretation$[0]       legs
application.lastResult$[1].interpretation$[1]       run
```

Privacy feature support for VoiceXML applications

In VoiceXML applications, you can declare a form or field to be private using the following property statement:

```
<property name="private" value="true"/>
```

While a private form or field is executing, Experience Portal does not write any speech recognition results, DTMF results, or TTS strings into the session transcription logs or into any alarms that may be generated during execution. Once the private form or field has finished executing, Experience Portal resumes logging these items as normal.

Note:

If tracing is turned on for the MPP, Experience Portal ignores the privacy property and writes the requested debugging information into the MPP trace logs.

Server Name Indication

The CCXML or VoiceXML browser examines the certificate that the server sends. It compares the name that the server is trying to connect to with the name included in the certificate. If the name matches, the browser continues the connection. Otherwise, it stops the connection.

DTMF digits sending by a VoiceXML application

Experience Portal allows VoiceXML applications to send DTMF digits. This feature is most commonly used when an application running on Experience Portal communicates with an automated system and not with a human being. This is also used when applications communicate with network routing, such as Cisco ICM for "tack back and transfer" applications. Previously, the only way for Experience Portal applications to send DTMF digits was to play an audio file that contained a recording of the digits to be sent.

Example

The following is a sample VoiceXML application that demonstrates how to send the digits 1 2 3 4:

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1"
xmlns="http://www.w3.org/2001/vxml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/vxml
http://www.w3.org/TR/2007/REC-voicexml21-20070619/vxml.xsd">
<form>
    <field>
        <prompt>
             <audio src="builtin://senddigit/1234"/>
             </prompt>
         </field>
```

```
    </form>
  </vxml>
```

CCXML elements and attributes

Elements	Attributes	Supporting information on Limitations and Extensions
<accept>	• connectionid • hints	For details on each specific instance of the **hints** attribute, see CCXML hints on page 27.
<assign>	• name • expr	If the CCXML compliance flag is set to **true**, the session variables like session* is checked for **read only**.
<cancel>	sendid	
<ccxml>	• version • xml:base • xsi:schemaLocation	For more information on the **xsi:schemaLocation** attribute, visit the following websites: • http://www.w3.org/2002/09/ccxml. • http://www.w3.org/TR/ccxml/ccxml.xsd.
<createcall>	• dest • connectionid • aai • callerid • hints • timeout • joinid • joindirection	For details on each specific instance of the **hints** attribute, see CCXML hints on page 27.

Table continues…

Elements	Attributes	Supporting information on Limitations and Extensions
<createccxml>	• next • namelist • fetchparam • parameters • method • sessionid • timeout • maxage • maxstale • enctype	
<createconference>	• conferenceid • confname • reservedtalkers • reservedlisteners • hints	
<destroyconference>	• conferenceid • hints	
<dialogprepare>	• src • type • namelist • parameters • dialogid • connectionid • conferenceid • mediadirection • maxage • maxstale • enctype • method • hints	For details on each specific instance of the **hints** attribute, see CCXML hints on page 27.

Table continues…

Elements	Attributes	Supporting information on Limitations and Extensions
<dialogstart>	• src • preparedialogid • type • namelist • parameters • dialogid • connectionid • conferenceid • mediadirection • maxage • maxstale • enctype • method • hints	• If the compliance flag is set to **true**, the flag checks if **maxage** is used with **preparedialogid**. In such a case, an error event is generated. • If the compliance flag is set to **true**, the flag checks if **maxstale**is used with **preparedialogid**. In such a case, an error event is generated. • If the compliance flag is set to **true**, the flag checks if **enctype** is used with **preparedialogid**. In such a case, an error event is generated. • If the compliance flag is set to **true**, the flag checks if **method** is used with **preparedialogid**. In such a case, an error event is generated. For details on each specific instance of the **hints** attribute, see CCXML hints on page 27.
<dialogterminate>	• dialogid • immediate • hints	EP does not support the **hints** attribute.
<disconnect>	• connectionid • reason • hints	
<else>		
<elseif>	cond	
<eventprocessor>	statevariables	

Table continues…

Elements	Attributes	Supporting information on Limitations and Extensions
<exit>	• expr • namelist	For EPM reporting, the following namelists are used: • avayaExitReason • avayaExitInfo1 • avayaExitInfo2 • avayaExitInfo5 • avayaExitInfo6 • avayaExitInfo7 • avayaExitInfo8 • avayaExitInfo9 • avayaExitPreferredPath • avayaExitCustomerId • avayaExitTopic
<fetch>	• next • type • namelist • method • fetchid • timeout • maxage • maxstale • timeout • mode	
<goto>	fetchid	
<if>	cond	

Table continues…

Elements	Attributes	Supporting information on Limitations and Extensions
<join>	• id1 • id2 • duplex • hints • entertone • exittone • autoinputgain • autooutputgain • dtmfclamp • toneclamp	For details on each specific instance of the **hints** attribute, see CCXML hints on page 27.
<log>	• label • expr	
<merge>	• connectionid1 • connectionid2 • hints	<Merge> is only supported by SIP, not by H.323.
<meta>	• name • http-equiv • content	
<metadata>		
<move>	• source • event • sessionid	
<redirect>	• connectionid • dest • reason • hints	For details on each specific instance of the **hints** attribute, see CCXML hints on page 27.
<reject>	• connectionid • reason • hints	For details on each specific instance of the **hints** attribute, see CCXML hints on page 27.

Table continues…

Elements	Attributes	Supporting information on Limitations and Extensions
<script>	• src • fetchid • timeout • maxage • maxstale • charset	• If the compliance flag is set to **true**, the src string should not include a single quote. In all other cases, the single quote is required. The following is an example of a noncompliant src string:`<script src="'http://<IP Address>:8080/FetchTestCCXML/jsp/updateaccount.js'"/>` The following is an example of a compliant src string:`script src="'http://<IP Address>:8080/FetchTestCCXML/jsp/updateaccount.js'"/>` • If the compliance flag is set to **true**, the charset string should not include a single quote. In all other cases, the single quote is required. The following is an example of a noncompliant charset string:`<script src=”script_utf_8.es”charset="'UTF-8'"/>` The following is an example of a compliant charset string: `<script src=”script_utf_8.es”charset="'UTF-8"/>`

Table continues…

Elements	Attributes	Supporting information on Limitations and Extensions
<send>	• target • targettype **Note:** The extra Avaya types are: - avaya_platform - avaya_plaform.web_service - SIPEndpoint • sendid • delay • name **Note:** If **name** is avaya.launchresponse, then **status** and **failed_status** can be used. The values for **status** and **failed_status** are: - success - no resource - busy - no answer - invalid telephony uri - network refuse - invalid application uri - appliction server failure - fax detected - network failed before connect - near end disconnect before connect - far end disconnect before connect - transferred before connect	For details on each specific instance of the **hints** attribute, see CCXML hints on page 27.

Table continues…

Elements	Attributes	Supporting information on Limitations and Extensions
	- unknown failure before connect - call disconnected before application started • namelist • hints	
<transition>	• state • event • cond	
<unjoin>	• id1 • id2 • hints	
<var>	• name • expr	

CCXML hints

<accept/>

Hint	Description
hints.enable_call_classification	Has the following values: • true • false Call classification is received in connection.signal events.
hints.call_classification_timeout	Specifies the timeout for call classification in milliseconds. For example, 20000 = 20 seconds.

<createcall/>

To change the ANI of a SIP call, set the callerid field. Hints are not needed. This does not work for H323 as there is no way to change the ANI for H323.

Hint	Description
hints.disable_cdr_logging	Disables CDR Logging on Voice Portal if the value is true.
hints.disable_video	Disables video on Voice Portal if the value is true.

Table continues…

Hint	Description
hints.enable_call_classification	Has the following values: • true • false Call classification is received in connection.signal events.
hints.call_classification_timeout	Specifies the timeout for call classification in milliseconds. For example, 20000 = 20 seconds.
hints.call_classification_recorded_msg_timeout	Specifies the timeout for call classification recorded message in milliseconds. For example, 20000 = 20 seconds. The timer starts after the call progress detects an answering machine greeting. That is the application receives a connection.signal event with the recorded_msg parameter. It is used to limit the amount of time that the call progress waits for the end of the answering machine message. If the timer fires, the application receives a connection.signal event with the session.connection.callprogress value set to timeout. The default is 30 seconds.
hints.sip.from.displayname	Sets the Display Name field in the From: header of the generated INVITE.
hints.sip.to.displayname	Sets the Display Name field in the To: header of the generated INVITE.
hints.sip.call-info	Sets the Call-Info: header in the INVITE.
hints.sip.organization	Sets the Organization: header in the INVITE.
hints.sip.subject	Sets the Subject: header in the INVITE.
hints.sip.priority	Sets the Priority: header in the INVITE.
hints.sip.replaces	Sets the Replaces: header in the INVITE.
hints.sip.x-nt-gslid	Adds a URL parameter named x-nt-gslid to the sip URI in the To: header and request line. This hint is used in propagating a GSLID on transfers and so on.
hints.sip.passertedid.displayname	Sets the Display Name field of the P-Asserted-Identity: header in the INVITE.
hints.sip.passertedid.uri	Sets the URI for the P-Asserted-Identity: header in the INVITE.

Table continues…

Hint	Description

hints.sip.historyinfo	Controls the history info headers in the generated INVITE request.
	`var hints = new Object();`
	`hints.sip = new Object();`
	`hints.sip.historyinfo = new Array();`
	`hints.sip.historyinfo[0] = new Object();`
	`hints.sip.historyinfo[0].user = "5551101";`
	`hints.sip.historyinfo[0].host = "avaya.com";`
	`hints.sip.historyinfo[0].index = "1";`
	`hints.sip.historyinfo[1] = new Object();`
	`hints.sip.historyinfo[1].displayname = "VDN1";`
	`hints.sip.historyinfo[1].user = "5551101";`
	`hints.sip.historyinfo[1].host = "avaya.com";`
	`hints.sip.historyinfo[1].optheader =`
	`"Reason=SIP;cause=302;text=\"Moved Temporarily\"";`
	`hints.sip.historyinfo[1].index = "1.1";`
	`hints.sip.historyinfo[2] = new Object();`
	`hints.sip.historyinfo[2].displayname = "Skill1";`
	`hints.sip.historyinfo[2].user = "5552101";`
	`hints.sip.historyinfo[2].host = "avaya.com";`
	`hints.sip.historyinfo[2].optheader =`
	`"Reason=SIP;cause=480;text=\"Temporarily Unavailable\"";`
	`hints.sip.historyinfo[2].index = "1.2";`

hints.sip.media	Controls whether an RTP stream should be negotiated for the call.
	It is used in SIP call scenarios like Best Service Routing (BSR) polling and call queueing that only require call signaling.
	`// Set media type empty so there is no media (rtp stream)`
	`var hints = new Object();`
	`hints.sip = new Object();`
	`hints.sip.media = new Array();`
	`hints.sip.media[0] = new Object();`
	`hints.sip.media[0].type = "empty";`

hints.sip.unknownhdr	Allows the setting of arbitrary headers in the generated INVITE.
	`var hints = new Object();`
	`hints.sip = new Object();`
	`hints.sip.unknownhdr = new Array();`
	`hints.sip.unknownhdr[0] = new Object();`
	`hints.sip.unknownhdr[0].name = "X-AnyHeader";`
	`hints.sip.unknownhdr[0].value = "Value for header";`

hints.call_classification_connectWhen	Has the following values: • OnConnect • OnProgress
hints.ConnectWhen	Has the following values: • OnConnected • OnProceeding • OnAlerting OnConnected is the default state if not specified. For more details, see ConnectWhen on page 31.

<dialogprepare/> or <dialogstart/>

Hint	Description
hints.ASRRequired	Overrides the setting in VPMS. The value could be true or false.
hints.ASRLanguages	Specifies the language used in ASR. For example, en-US or en-SG. **Note:** Check VPMS on Speech Servers for the list of all languages.

<join/>

Hint	Description
hints.clamp_dtmf_duplex	Blocks DTMF (DTMF clamping) in either full duplex or half duplex direction. It is used only when dtmfclamp join parameter is `true`. Has the following values: • half: Clamps DTMF from id1 to id2, and allows DTMF from id2 to id1. • full: Clamps DTMF in both directions.

The following is an example of using clamp dtmf half duplex:

```
< join id1="confid" id2="agentid" dtmfclamp="'true'" duplex="'full'"
hints="{clamp_dtmf_duplex:'half'}"/>
```

This blocks DTMF from conference to agent, but allows DTMF from agent to conference.

<merge/>

Hint	Description
hints.MergeTimeout	Sets a time limit for the merge to complete, specified in milliseconds. If the underlying REFER w/Replaces operation doesn't complete within the time limit, the merge will fail.
hints.DestinationURI	Sets the URI in the ReferTo: header of the REFER w/Replaces request.

<redirect/>

Hint	Description
hints.ConnectWhen	Contains the following values: • OnConnected • OnProceeding • OnAlerting OnProceeding is the default state if not specified. For more details, see ConnectWhen on page 31.
hints.TransferTimeout	Indicates the time interval of finishing the transfer.

Table continues…

Hint	Description
hints.AAI	Specifies the application to application info. AAI is a string containing data sent to an application on the far-end, available in the session variable session.connection.aai. For details on the <transfer/> element, see VoiceXML elements and attributes on page 33.

<reject/>

Hint	Description
hints.sip.respcode	Sets the numeric value of the SIP status code. This allows the application to override the default of 302.
hints.sip.resptext	Sets the string value of the SIP reason phrase. This allows the application to override the default of Moved Temporarily.

For more details, see Sample method for passing AAI as UUI on page 32.

<send/>

Hint	Description
hints.h323	Specifies H323 call info.
hints.uui	Sets UUI info.
session.connection.protocol.sip.requestmethod	Contains the following values: • info • options • update These three values can be used with targettype "SIPEndpoint" on <send/> to determine the SIP request message.
session.connection.protocol.sip.body[xx].type	Specifies the content type for the multipart message body of the SIP request.
session.connection.protocol.sip.body[xx].msg	Specifies the content for the multipart message body of the SIP request.

For more details, see Sample method for sending a SIP INFO message on page 32.

ConnectWhen

ConnectWhen is a field that indicates when a transfer should be considered complete.

ConnectWhen only works in the redirect tag and when the call that is being redirected is in the connected state. The connected state is where a transfer can be performed.

ConnectWhen contains the following values:

- OnConnected: OnConnected is a supervised transfer, where the transfer is not considered successful until the outgoing call is answered. The ConnectWhen hint will be ignored if it is used in any other context.

- OnProceeding: OnProceeding tells the platform to consider the transfer successful when the outgoing call reaches the proceeding state. The proceeding state is where the switch has accepted the transfer. This is called a blind transfer.
- OnAlerting: OnAlerting tells the platform to consider the transfer successful when the outgoing call reaches the alerting state. The alerting state is where the call rings. This is a transfer to a ringing type operation.

Note:

On createcall, ccxml passes the ConnectWhen hint to Session Manager, however Telephony might ignore it.

Sample method for sending a SIP INFO message

The following is an example for sending a SIP INFO message:

```
<transition event="dialog.exit"  state="in_dialog">
        <log expr="'-- ' + event$.name +' -- [' + state +']'"/>
        <log expr="'   eventdata... \n' + objectToString(event$)"/>
        <!--
         Save return values and log
         -->
        <assign name="dialogresult" expr="event$.values.dialogresult" />
        <assign name="collecteddigits" expr="event$.values.collecteddigits" />
        <log expr="'  dialogresult : ' + dialogresult"/>
        <log expr="'  collecteddigits : ' + collecteddigits"/>
        <if cond="dialogresult == 'CANCEL'">
         <log expr="'User quit application exiting'" />
         <disconnect connectionid="in_connectionid"/>
        </if>

        <!-- Construct a SIP INFO message to send the collected digits -->
        <script>
     var hints = new Object();
     hints.sip = new Object();
     hints.sip.requestmethod = 'INFO';
     hints.sip.body = new Array(1);
     hints.sip.body[0] = new Object();
     hints.sip.body[0].type = 'application/vnd.nortelnetworks.digits';
     hints.sip.body[0].msg = 'p=Digit-Collection\ny=Digits\nd=Digits%3D'+
collecteddigits;
 </script>
 <send name="''" target="in_connectionid" targettype="'SIPEndpoint'" hints="hints"/>

        <log expr="'Program quit application exiting'" />
        <disconnect connectionid="in_connectionid"/>
    </transition>
```

Sample method for passing AAI as UUI

The following is an example for passing AAI as UUI:

```
    <transition state="in_dialog init incoming_sip" event="dialog.transfer">
<var name="dialog_values" expr="event$.values"/>
<assign name="dialog_values.AAI" expr="event$.aai"/>
      <redirect connectionid="main_connectionid" dest="event$.uri"
hints="dialog_values"/>
    </transition>
```

Note:

If the customer is using default.ccxml in a SIP environment, then the hints to pass AAI as UUI will already be applied if using a VXML transfer.

VoiceXML elements and attributes

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<assign>	• name • expr	Yes	Yes		Supports all attributes.
<audio>	• src • fetchtimeout • fetchhint • maxage • maxstale • expr	Yes	Yes		Supports all other attributes except **fetchhint**.
<block>	• name • expr • cond	Yes	Yes		Supports all attributes.
<catch>	• event • count • cond	Yes	Yes		Supports all attributes.

Table continues…

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<choice>	• dtmf • accept • next • expr • event • eventexpr • message • messageexpr • fetchaudio • fetchhint • fetchtimeout • maxage • maxstale	Yes	Yes		Supports all attributes except **message**, **messageexpr**, and **fetchhint**.
<clear>	• namelist	Yes	Yes		Supports the **namelist** attribute.
<initial>	• name • expr • cond	Yes	Yes		Supports all attributes.
<disconnect>	• namelist	No	Yes		Supports the **namelist** attribute.
<else>	• name • expr • cond	Yes	Yes		Supports all attributes.
<error>	count cond	Yes	Yes		Supports all attributes.
<exit>	expr namelist	Yes	Yes		Supports all attributes.

Table continues…

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<field>	• name • expr • cond • type • slot • modal	Yes	Yes		Supports all attributes.
<filled>	• mode • namelist	Yes	Yes		Supports all attributes.
<form>	• id • scope	Yes	Yes		Supports all attributes.
<goto>	• next • expr • nextitem • expritem • fetchaudio • fetchint • fetchtimeout • maxage • maxstale	Yes	Yes		Supports all attributes.

Table continues…

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<grammar>	• version • xml:lang • mode • root • tag-format • xml:base • src • scope • type • weight • fetchhint • fetchtimeout • maxage • maxstale • srcexpr	Yes	Yes		Supports all attributes except **fetchhint**.
<help>	• count • cond	Yes	Yes		Supports all attributes.
<if> (optional <else> and <elseif> elements)		Yes	Yes		Supports the **if** element.

Table continues…

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<link>	• next • expr • eventexpr • message • messageexpr • dtmf • fetchaudio • fetchhint • fetchtimeout • maxage • maxstale	Yes	Yes		Supports all attributes except **fetchhint**.
<log>	• label • expr	Yes	Yes		Supports all attributes.
<mark> (an SSML element)	• name • nameexpr • markname • marktime	• name — Yes • nameexpr — No • markname — No • marktime — No	Yes		Supports **name** and **nameexpr** only. **Note:** In VoiceXML 2.0, the ,<mark> element was ignored by VoiceXML platforms.
<media>	• src • srcexpr • clipBegin • clipEnd • repeatDur • repeatCount	No	No	Yes	soundLevel not supported.

Table continues…

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<menu>	• id • scope • dtmf • accept	Yes	Yes		Supports all attributes.
<meta>	• name • content • http-equiv	Yes	Yes		Currently inactive.
<metadata>	• creator • rights • subject	Yes	Yes		Currently inactive.
<noinput>	• count • cond	Yes	Yes		Supports all attributes.
<nomatch>	• count • cond	Yes	Yes		Supports all attributes.
<object>	• name • expr • cond • classid • codebase • codetype • data • type • archive • fetchhint • fetchtimeout • maxage • maxage	Yes	Yes		Currently inactive.
<option>	• dtmf • accept • value	Yes	Yes		Currently inactive.

Table continues…

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<param>	• name • expr • value • valuetype • type	Yes	Yes		• Currently inactive — **valuetype** and **type**. • Supports all the other attributes.
<prompt>	• bargein • bargeintype • cond • count • timeout • xml:lang • xml:base	Yes	Yes		Supports all attributes.
<property>	• name • value	Yes	Yes		Supports all attributes.
<record>	• name • expr • cond • modal • beep • maxtime • finalsilence • dtmfterm • type	Yes	Yes		Supports all attributes.
<reprompt>		Yes	Yes		Supports the <reprompt> element.
<return>	• event • eventexpr • message • messageexpr • namelist	Yes	Yes		Supports all attributes.

Table continues…

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<script>	• src • charset • fetchhint • fetchtimeout • maxage • maxstale • srcexpr	• srcexpr — No • Rest of the attributes— Yes	Yes		Supports all attributes except **fetchhint**.
<subdialog>	• name • expr • cond • namelist • srcexpr • method • enctype • fetchaudio • fetchtimeout • maxage • maxstale	Yes	Yes		Supports all attributes.
<submit>	• next • expr • namelist • method • enctype • fetchaudio • fetchhint • fetchtimeout • maxage • maxstale	Yes	Yes		Supports all attributes except **fetchhint**.

Table continues…

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<throw>	• event • eventexpr • message • messageexpr	Yes	Yes		Supports all attributes.
<transfer>	• name • expr • cond • dest • destexpr • bridge • connecttimeout • maxtime • transferaudio • aai • aaiexpr • type	• type — No • Rest of the attributes — Yes	Yes		Supports all attributes.
<value>	expr	Yes	Yes		Supports the attribute.
<var>	• name • expr	Yes	Yes		Supports all attributes.
<vxml>	• version • xmlns • xml:base • xml:lang_d • application	Yes	Yes		Supports all attributes.

Table continues…

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<data>	• src • name • srcexpr • method • namelist • enctype • fetchaudio • fetchhint • fetchtimeout • maxage • maxstale	No	Yes		Supports all attributes except fetchhint.
<foreach>	• array • item	No	Yes		Supports all attributes.
<receive>	• fetchaudio • fetchaudioexpr • maxtime • maxtimeexpr	No	No	Yes	Only works from launching the CCXML application to the launched application.

Table continues…

Tag	Attributes	Available in VoiceXML 2	Available in VoiceXML 2.1	Available in VoiceXML 3.0	Limitation or Extensions, Supporting information
<send>	• async • asyncexpr • body • bodyexpr • contenttype • contenttypeexpr • event • eventexpr • fetchaudio • fetchaudioexpr • namelist • timeout • timeoutexpr	No	No	Yes	Only works from launching the CCXML application to the launched application. The target and targetexpr attributes must not be specified.

Chapter 4: Call classification in speech applications

Call classification overview

Call classification, or call progress, is a method for determining who or what is on the other end of a call by analyzing the audio stream. When the analysis determines a probable match, the CCXML page receives a `connection.signal` event which contains the result in the `event$.info.callprogress` variable. Depending on the results of the analysis, call classification may continue for the duration of the call. In that case, the CCXML page will may receive multiple `connection.signal` events.

Call classification falls into two categories:

- Tone-based classification. This category has a high degree of accuracy because it is easy for the system to detect busy signals or fax machine signals.
- Speech-based classification. This category as a much lower degree of accuracy because it can be difficult to tell a human being's speech from an answering machine's recorded message.

Experience Portal uses speech-based classification when it detects an amount of energy in a frequency consistent with human speech. When it detects human speech, it differentiates between a live human being and an answering machine based on the length of the utterance. Generally, if a live human being answers, the utterance is relatively short while answering machine greetings are usually much longer.

For example, a live human might just say “Hello.” while an answering machine message might say “Hello. I can't come to the phone right now. Please leave your message after the beep.”

However, some people may answer the phone with a much longer greeting while others have recorded a very short answering machine greeting. In both of these cases, the call classification algorithm will incorrectly identify the call and assume the long greeting is a recorded message while the short greeting is a live human being.

To use call classification in your applications, make sure you design the application so that it takes such mistakes into account.

If you configure Experience Portal system to use H.323 connections and outcall applications, you must have Communication Manager with the SA8874 feature to detect that the call has been answered. Without the SA8874 feature, Experience Portal incorrectly detects that the call is answered even if the call is not answered.

If you configure Experience Portal system to use outcall applications, and want the Experience Portal system to differentiate human vs. answering machine, you require the Enhanced Call Classification license.

Call classification analysis results

When Experience Portal classifies the call, it sends a `connection.signal` event to the CCXML page. This event includes the results of the classification in the `event$.info.callprogress` variable.

Value of `event$.info.callprogress`	Applies to	Description
`start_of_voice`	Outbound calls only	The call is answered and Experience Portal detects a voice on the line. EP determines whether the voice is from a human being or from an answering machine. A `Start_of_voice` classification is followed by either a `live_voice` or `recorded_msg` classification.
`live_voice`	Outbound calls only	The call is probably connected to a human being. No further classifications will be sent for this call.
`busy_tone`	Outbound calls only	A busy tone was received. This is commonly referred to as the “slow busy” tone. No further classifications will be sent for this call.
`reorder`	Outbound calls only	A switch error, such as all circuits being busy, has occurred. This is commonly referred to as the “fast busy” tone. No further classifications will be sent for this call.
`sit_tone`	Outbound calls only	A special information tone was received which indicates that the call could not be completed. This is the three frequency tone that is often followed by a spoken message. No further classifications will be sent for this call.
`recorded_msg`	Outbound calls only	An answering machine was detected and a recorded message has just started. A `recorded_msg` classification will always be followed by either a `msg_end` or `timeout` classification. No further classifications will be sent for this call.
`msg_end`	Outbound calls only	The end of a recorded message, such as that played by an answering machine, was detected. No further classifications will be sent for this call.

Table continues…

Value of `event$.info.callprogress`	Applies to	Description
`fax_calling_tone`	Inbound calls only	A fax machine was detected as the initiator of an inbound call. No further classifications will be sent for this call.
`fax_answer_tone`	Outbound calls only	A fax machine was detected as the recipient of an outbound call. No further classifications will be sent for this call.
`ringing`	Outbound calls only	A “ring back” tone was detected. One or more of these classifications may be received by the application, but they are always followed by one of the other applicable classifications.
`timeout`	Inbound and outbound calls	The classification algorithm failed to classify the call before the allotted time ran out. By default, the allotted time is 20 seconds (or 20000 milliseconds). The default value can be overridden for outbound calls by setting the `call_classification_timeout` parameter in the `hints` attribute on the `<createcall>` tag to the desired number of milliseconds before call classification analysis should time out. No further classifications will be sent for this call.
`error`	Inbound and outbound calls	An internal error occurred during the classification analysis and the call could not be properly classified. No further classifications will be sent for this call.
early_media	Outbound calls only	Early media refers to media that is exchanged before a particular session is accepted by the called user. For example, for outbound calls the color ring tone will be detected as early_media by call classification.

Call classification for inbound calls

The only call classification provided for inbound calls is the ability to detect an incoming fax. It is extremely difficult to differentiate between a live human being and a recorded message for an incoming call because you cannot assume the initial greeting will be as short for an incoming call as it is for an outgoing call. Therefore, any classification for an incoming call other than `fax_calling_tone` should be treated as if a live human being was detected.

If fax tone is detected when using VoiceXML applications, Experience Portal stops the application and transfers the call to a fax machine.

If fax tone is detected when using CCXML applications, Experience Portal sends the fax_calling_tone event to the application. The call treatment is then decided by the rules which the application designer has configured on the application.

When you add an application to Experience Portal using the EPM web interface, the following parameters in the **Advanced Parameters** group on the Add Application page determine what happens when an incoming fax machine call is detected:

Parameter	Description
Fax Detection Enabled	The options are: • **Yes**: The application should attempt to identify whether the caller is a fax machine and route any fax machine calls to the telephone number specified in **Fax Phone Number**. • **No**: The application should not attempt to identify whether the caller is a fax machine. The default is **No**.
Fax Phone Number	If **Fax Detection Enable** is set to **Yes**, this is the telephone number or URI to which fax machines calls should be routed.

Call classification for outbound calls

Classification of human vs. answering machine for outbound call is done by Experience Portal. The classification is based on the length of the voice energy that is detected after the call is answered. You require the Enhanced Call Classification license on the Experience Portal system to differentiate human vs. answering machine. The classification is done only if the application sends a classification request to Experience Portal.

Note:

In the VoiceXML <TRANSFER> outcall method, the human vs. answering machine classification is never done.

Detection of answer, busy, RNA, and so on classification is done by Communication Manager and not by Experience Portal. Communication Manager needs the SA8874 feature to detect this classification. The detection is done on all outcalls except for blind transfers.

The application designer needs to enable call classification for outbound calls. If the call is going to be invoked by the Application Interface web service `LaunchVXML` method, you can specify the call classification parameters when you invoke the web service, as described in Call classification with the LaunchVXML method on page 48.

Otherwise, the application designer has to set `enable_call_classification=true` in the `hints` attribute of the `<createcall>` tag.

When call classification is enabled, a call will receive one or more `connection.signal` events containing the `event$.info.callprogress` field. This field will have one of the values described in Call classification for outbound calls on page 47.

Note:

Remember that the page may receive `connection.signal` events that do *not* contain the `callprogress` field. It is up to the page to determine if the`callprogress` field exists and to take the appropriate course of action based on the value of this field.

The default timeout for outbound call classification is 20 seconds (or 20000 milliseconds). The default value can be overridden for outbound calls by setting the `call_classification_timeout` parameter in the `hints` attribute on the `<createcall>` tag to the desired number of milliseconds before call classification analysis should time out.

The following example shows a simple `connection.signal` handler page that first determines whether this is a `connection.signal` event bearing classification data. If it is, the handler assesses the data to determine what action to take. If the classification is `live_voice`, then it returns a status of `success` and the page continues to run. Otherwise, a it returns the failure status `no answer`.

```
<transition event="connection.signal">
    <if cond="typeof(event$.info) != 'undefined'">
        <if cond="typeof(event$.info.callprogress) != 'undefined'">
            <var name="call_classification" expr="event$.info.callprogress"/>
            <var name="status"/>
            <if cond="call_classification == 'live_voice'">
                <assign name="status" expr="'success'"/>
                <send name="'avaya.launchresponse'" targettype="'avaya_platform'"
                        target="session.id" namelist="status"/>
            <else/>
                <assign name="status" expr="'no answer'"/>
                <send name="'avaya.launchresponse'" targettype="'avaya_platform'"
                        target="session.id" namelist="status"/>
            </if>
        </if>
    </if>
</transition>
```

Call classification with the LaunchVXML method

Call classification allows the VoiceXML application to return the appropriate status code based on whether a human, an answering machine, or a fax machine answers an outbound call.

Call classification parameters for the LaunchVXML method

The following call classification name-value pairs can be passed as parameters with the LaunchVXML method. For both parameters the default is `false`, which means that you must specify the name-value pair in order to enable the associated functionality.

Name-value pair	Description
`enable_call_classification=true`	This required parameter enables call classification.
`detect_greeting_end=true`	This optional parameter instructs the VoiceXML application to identify the end of a recorded greeting if an answering machine answers the outbound call.
`call_classification_recorded_msg_timeout = in mili secs (e.g. 30000 is 30 sec)`	This Optional parameter is to set wait timeout for “end of recorded greeting”, if an answering machine answers the outbound call. Default is 30 sec.
`call_classification_connectWhen = ( OnConnect or OnProgress)`	This optional parameter is to start the CPA engine (call classification) on either OnConnect or OnProgress. By default it is set to OnProgress. In case the engine is started before connect, early media will also be captured for call classification.
`call_classification_timeout=value`	Timeout for outbound call classification from engine. This optional parameter indicates how long the call classification function will run if it is unable to determine the classification. The value specified should be in milliseconds. If the value is not provided, the default timeout is 20 sec.

Call classifications

If you enable call classification, the VoiceXML application sends one of the following classifications to the application server using the query arguments on the URL:

Classification	Description
`live_voice`	If a human being answers the call, the application starts the previously-prepared VoiceXML dialog. **Note:** This is the default classification assigned to the VoiceXML session before the call is placed. If a human being does not answer the call, this classification must be changed.
`recorded_msg`	If the LaunchVXML method was invoked with `detect_greeting_end=false` or if the `detect_greeting_end` parameter was not specified and an answering machine answers the call, the application terminates the previously-prepared VoiceXML dialog and starts a new dialog by sending the classification `recorded_msg` to the application server.
`msg_end`	If the LaunchVXML method was invoked with `detect_greeting_end=true` and an answering machine answers the call, the application terminates the previously-prepared VoiceXML dialog and starts a new dialog by sending the classification `msg_end` to the application server.
`fax_answer_tone`	If a fax machine answers the call, the application terminates and returns the error code `fax detected (8206)` to the Application Interface web service.

Table continues…

Classification	Description
`timeout`	If the VoiceXML application does not send a classification change message to the CCXML page within a given period of time, the CCXML applications assumes that a live person has answered the phone and it starts the previously-prepared VoiceXML dialog.
`*`	All other classifications result in the status code for `no answer ()` being returned the Application Interface web service.

Chapter 5: SIP application support

User-to-User Interface (UUI) data passed in SIP headers

When you add an application to Experience Portal using the EPM web interface, you also specify how User-to-User Interface (UUI) information will be passed to the application if it uses a SIP connection using the **Operation Mode** field in the **Advanced Parameters** group on the Add Application page.

The options are:

- Service Provider: Experience Portal passes the UUI data along as a single block to the application without making any attempt to interpret data.

  If you select this option, the application must handle the UUI data on its own.

- Shared UUI: Experience Portal takes the UUI data and parses it into an array of IDs and their corresponding values. It then passes the application both the fully encoded UUI data and the parsed array with only the values still encoded..

  If you select this option, the UUI data must conform to the Avaya UUI specifications listed below.

UUI data format in CCXML applications

For a CCXML application, the UUI data is organized in a tree format. For example:

```
avaya.ucid = '####################'
avaya.uui.mode = "shared uui"
avaya.uui.shared[0].id = "FA"
avaya.uui.shared[0].value = "#################"
avaya.uui.shared[1].id = "BG"
avaya.uui.shared[1].value = "#################"
```

Each connection has its own tree associated with it, and the values for that connection can be accessed using the format `session.connections['`*connection_number*`'].avaya.`*element_name*.

For example, if you want to declare two variables called `ucid` and `mode`, and you wanted to set those variables to be equal to the corresponding values for connection 1234, you would specify:

```
<var name="ucid" expr="session.connections['1234'].avaya.ucid"/>
<var name="mode" expr="session.connections['1234'].avaya.uui.mode"/>
```

UUI data format in VoiceXML applications

In VoiceXML, there is only one active call so the UUI information is organized into a similar tree format and placed into a session variable at the start of the dialog. The tree structure looks like this:

```
session.avaya.ucid
session.avaya.uui.mode
session.avaya.uui.shared[]
```

With the Shared UUI mode, you must send UUI/AAI data as name-value pairs in the following format:

```
<id0>,<value0>;<id1>,<value1>;<id2>,<value2>; ...
```

When you specify the name-value pairs:

- Each name must be set to the hexadecimal encoded ID stored in the `shared[]` array.
- Each value must be set to the encoded value stored in the `shared[]` array.
- Each name in the pair must be separated from its value by a `,` (comma).
- Each name-value pair must be separated from the next name-value pair by a `;` (semi-colon).

For example, if you wanted to send a UCID using UUI/AAI, you might specify: `aai = "FA,2901000246DEE275"`

Universal Call Identifier (UCID) values included in UUI data

The Universal Call Identifier (UCID) is an Avaya-proprietary call identifier used to help correlate call records between different systems. This identifier can either be generated by the Experience Portal MPP server or it can be passed to Experience Portal through an application's SIP headers if the application uses a SIP connection and the application's **Operation Mode** is set to **Shared UUI**.

Note:

If the application uses an H.323 connection, Experience Portal can receive UCID from Communication Manager. This feature is supported in Communication Manager 5.2.

To enable this feature, you need to administer ucid-info on button 10 on the 7434ND stations used by Experience Portal.

A UCID consists of 20 decimal digits in three groups. Before the UCID is added to the UUI data, Experience Portal encodes each group as a hexadecimal value and concatenates the three groups together. The:

- First group of 5 digits represents a 2 byte network node identifier assigned to the Communication Manager. In the UUI data, this group is encoded as 4 hexadecimal digits.
- Second group of 5 digits represents a 2 byte sequence number. This group is encoded as 4 hexadecimal digits.
- Third group of 10 digits represents a 4 byte timestamp. This group is encoded as 6 hexadecimal digits.

For example, the UCID 00001002161192633166 would be encoded as `000100D84716234E`.

When Experience Portal passes the UCID 00001002161192633166 to the application, it would look like this:

```
avaya.ucid = '00001002161192633166'
avaya.uui.mode = 'shared uui'
avaya.uui.shared[0].id = 'FA'
avaya.uui.shared[0].value = '000100D84716234E'
```

Note:

The identifier for the UCID is always 250, which becomes FA in hexadecimal in the UUI `shared[]` array.

Experience Portal application parameters affecting the UUI data

When you add an application to Experience Portal using the EPM web interface, the following parameters in the **Advanced Parameters** group on the Add Application page affect the contents of the SIP UUI data:

Parameter	Description
Generate UCID	The Universal Call Identifier (UCID) is an Avaya-proprietary call identifier used to help correlate call records between different systems. The options are: • **Yes**: If the CM does not pass a UCID to Experience Portal, the MPP server generates a UCID. • **No**: The MPP does not generate a UCID.
Operation Mode	The SIP header for each call can contain User-to-User Interface (UUI) information that the switch can either pass on as a single block or attempt to parse so that the information can be acted on. This field determines how Experience Portal treats the UUI data. The options are: • **Service Provider**: Experience Portal passes the UUI data as a single block to the application without making any attempt to interpret data. If you select this option, the application must handle the UUI data on its own. • **Shared UUI**: Experience Portal takes the UUI data and parses it into an array of IDs and their corresponding values. It then passes the application both the fully encoded UUI data and the parsed array with only the values still encoded. If you select this option, the UUI data must conform to the Avaya UUI specifications described in User-to-User Interface (UUI) data passed in SIP headers on page 51.

Table continues…

Parameter	Description
Transport UCID in Shared Mode	If **Operation Mode** is set to **Shared UUI** and **Generate UCID** is set to **Yes**, this field determines whether Experience Portal encodes the Experience Portal-generated UCID and adds it to the UUI data for all outbound calls. The default is **No**, which means that a UCID is only passed as part of the UUI information if that UCID was passed to Experience Portal by the application.
Maximum UUI Length	The maximum length of the UUI data that can be passed in the SIP header. If this length is exceeded and Experience Portal generated a UCID for the call, the UCID is removed from the UUI data. If the result still exceeds this value, or if the UCID was passed to Experience Portal by the application, Experience Portal does *not* send any UUI data. Instead, it leaves the entire field blank because it has no way to determine what can be left out.

SIP header support for CCXML and VoiceXML applications

Session Initiation Protocol (SIP) headers can provide additional information about a call that a CCXML or VoiceXML application can use to determine what processing needs to be done for that call. Experience Portal uses a collection of session variables to present this information to the application. However, not all SIP headers are accessible and many accessible headers are read only.

The following table lists the session variables that may accompany an inbound SIP INVITE. In the table, `<sip>` is the variable access string used to access the variables in a particular context. The valid strings are:

Variable access string	Description
`session.connection.protocol.sip`	This access string can be used by either CCXML or VoiceXML applications.
`event$.info.protocol.sip`	This access string can used when variables arrive in the `info` map for a transition. These variables are only valid within the transition in which they arrived.
`session.connections['`*`SessionID`*`'].protocol.sip` where *SessionID* is the session ID.	This access string can be used to retrieve the variables from the connection object in the session variable space. The variables exist between transitions but can be overwritten by new data at any time.

If you want to use a variable in your application, you must use the complete text in your code.

For example, if you want to access the `<sip>.callid` variable in a session with the session ID *1234*, you would code one of the following, depending on the context in which you want to access the variable:

- `session.connection.protocol.sip.callid`
- `event$.info.protocol.sip.callid`
- `session.connections['1234'].protocol.sip.callid`

Session Variable	SIP Header	Notes
<sip>.callid	Call-ID	Uniquely identifies a particular invitation or all registrations of a particular client.
<sip>.contact[array].displayname <sip>.contact[array].uri	Contact	Provides the display name, a URI with URI parameters, and header parameters.
<sip>.from.displayname <sip>.from.uri <sip>.tag	From	The initiator of the request.
<sip>.historyinfo[array].displayname <sip>.historyinfo[array].user <sip>.historyinfo[array].host <sip>.historyinfo[array].optheader	History-Info	This field is typically used to inform proxies and User Agent Clients and Servers involved in processing a request about the history or progress of that request.
<sip>.passertedid[array].displayname <sip>.passertedid[array].uri	P-asserted Identity	The verified identity of the user sending the SIP message. This field is typically used among trusted SIP intermediaries to prove that the initial message was sent by an authenticated source.
<sip>.require	Require	The options that the User Agent Client (UAC) expects the User Agent Server (UAS) to support in order to process the request.
<sip>.supported[array].option	Supported	All extensions supported by the UAC or UAS.
<sip>.to.displayname <sip>.to.host <sip>.to.uri <sip>.to.user	To	The logical recipient of the request.

Table continues…

Session Variable	SIP Header	Notes
<sip>.unknownhdr[array].name <sip>.unknownhdr[array].value	Unknown	All headers not understood by Avaya Experience Portal are passed to the application through this array.
<sip>.useragent[array]	User-Agent	Contains information about the UAC originating the request.
<sip>.via[array].sentaddr <sip>.via[array].sentport <sip>.via[array].protocol <sip>.via[array].branch	Via	The path taken by the request to this point along with the path that should be followed in routing responses.
<sip>.name		This variable returns “sip” when the SIP protocol is used.
<sip>.version		This variable returns the SIP protocol version when the SIP protocol is used.
<sip>.requestmethod <sip>.requestversion <sip>.requesturi <sip>.request.user <sip>.request.host <sip>.requestparams[array].name <sip>.requestparams[array].value	“request”	The various components of the request URI (INVITE). For example, this variable includes the parameters, user and host part of the URI, and the request method.
<sip>.respcode		The results of a transaction. The actual contents varies by transaction type.
<sip>.resptext		The results of a transaction. The actual contents varies by transaction type.

Sample VoiceXML page logging SIP headers

The following VoiceXML page logs various SIP headers using the Experience Portal session variables.

```
<?xml version="1.0"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml" xml:lang="en-us">
<property name="promptgender" value="female"/>
<property name="timeout" value="4s"/>
<property name="maxspeechtimeout" value="15s"/>
 <form id="form0">
  <block>
    <log>session.connection.protocol.sip <value
    expr="session.connection.protocol.sip"/></log>
```

```
    <log>session.connection.protocol.name: <value
    expr="session.connection.protocol.name"/></log>
    <log>session.connection.protocol.version: <value
    expr="session.connection.protocol.version"/></log>
    <log>session.connection.protocol.sip.requesturi: <value
    expr="session.connection.protocol.sip.requesturi"/></log>
    <log>session.connection.protocol.sip.requestmethod: <value
    expr="session.connection.protocol.sip.requestmethod"/></log>
    <log>session.connection.protocol.sip.requestversion: <value
    expr="session.connection.protocol.sip.requestversion"/></log>
    <log>session.connection.protocol.sip.request.user: <value
    expr="session.connection.protocol.sip.request.user"/></log>
    <log>session.connection.protocol.sip.request.host: <value
    expr="session.connection.protocol.sip.request.host"/></log>
    <log>session.connection.protocol.sip.requestparams[0].name: <value
    expr="session.connection.protocol.sip.requestparams[0].name"/></log>
    <log>session.connection.protocol.sip.requestparams[0].value: <value
    expr="session.connection.protocol.sip.requestparams[0].value"/></log>
    <log>session.connection.protocol.sip.to.uri: <value
    expr="session.connection.protocol.sip.to.uri"/></log>
    <log>session.connection.protocol.sip.to.displayname: <value
    expr="session.connection.protocol.sip.to.displayname"/></log>
    <log>session.connection.protocol.sip.to.user: <value
    expr="session.connection.protocol.sip.to.user"/></log>
    <log>session.connection.protocol.sip.to.host: <value
    expr="session.connection.protocol.sip.to.host"/></log>
    <log>session.connection.protocol.sip.from.uri: <value
    expr="session.connection.protocol.sip.from.uri"/></log>
    <log>session.connection.protocol.sip.from.displayname: <value
    expr="session.connection.protocol.sip.from.displayname"/></log>
    <log>session.connection.protocol.sip.from.tag: <value
    expr="session.connection.protocol.sip.from.tag"/></log>
    <log>session.connection.protocol.sip.from.user: <value
    expr="session.connection.protocol.sip.from.user"/></log>
    <log>session.connection.protocol.sip.from.host: <value
    expr="session.connection.protocol.sip.from.host"/></log>
    <log>session.connection.protocol.sip.useragent[0]: <value
    expr="session.connection.protocol.sip.useragent[0]"/></log>
    <log>session.connection.protocol.sip.contact[0].displayname: <value
    expr="session.connection.protocol.sip.contact[0].displayname"/></log>
    <log>session.connection.protocol.sip.contact[0].uri: <value
    expr="session.connection.protocol.sip.contact[0].uri"/></log>
    <log>session.connection.protocol.sip.via[0].sentaddr: <value
    expr="session.connection.protocol.sip.via[0].sentaddr"/></log>
    <log>session.connection.protocol.sip.via[0].protocol: <value
    expr="session.connection.protocol.sip.via[0].protocol"/></log>
    <log>session.connection.protocol.sip.via[0].sentport: <value
    expr="session.connection.protocol.sip.via[0].sentport"/></log>
    <log>session.connection.protocol.sip.via[1].sentaddr: <value
    expr="session.connection.protocol.sip.via[1].sentaddr"/></log>
    <log>session.connection.protocol.sip.via[1].protocol: <value
    expr="session.connection.protocol.sip.via[1].protocol"/></log>
    <log>session.connection.protocol.sip.via[1].sentport: <value
    expr="session.connection.protocol.sip.via[1].sentport"/></log>
    <log>session.connection.protocol.sip.supported: <value
    expr="session.connection.protocol.sip.supported"/></log>
    <log>session.connection.protocol.sip.require: <value
    expr="session.connection.protocol.sip.require"/></log>
  </block>
 </form>
</vxml>
```

Support for unknown headers

If Experience Portal receives an INVITE with a header it does not recognize, it saves the name and value of the header in the `session.connection.protocol.sip.unknownhdr` session variable array.

For example, if Experience Portal receives an INVITE with the following unknown headers:

```
new_header: "helloworld"
another_new_header: "howareyou"
```

Experience Portal adds the following entries to the `session.connection.protocol.sip.unknownhdr` array:

```
session.connection.protocol.sip.unknownhdr.unknownhdr[0].name = "new_header"
session.connection.protocol.sip.unknownhdr.unknownhdr[0].value = "helloworld"
session.connection.protocol.sip.unknownhdr.unknownhdr[1].name = "another_new_header"
session.connection.protocol.sip.unknownhdr.unknownhdr[1].value = "howareyou"
```

RFC 3261 SIP headers

You can set a limited number of SIP headers independently for applications that initiate an outbound call with one of the following:

- A REFER as a result of a blind or consultative transfer
- An INVITE as a result of bridged transfer or a CCXML outbound call

Note:

Not all headers that are available to be set on an INVITE are available to be set on a REFER.

You can set the following headers with the VoiceXML `<property>` element before `<transfer>` element. In the table, `<sip_prop>` represents `AVAYA_SIPHEADER.session.connection.protocol.sip`.

Note:

If you want to set a header in your application, you must use the complete text in your code. For example, code `<sip_prop>.callid` as `AVAYA_SIPHEADER.session.connection.protocol.sip.callid`.

SIP Header	<property>	Notes
Call-Info	<sip_prop>.callinfo	Provides additional information about the source or target of the call, depending on whether it is found in a request or response.
To.displayname	<sip_prop>.to.displayname	Displays the name of the call's target.

Table continues…

SIP Header	<property>	Notes
From.displayname	<sip_prop>.from.displayname	Displays the name of the call's source.
P-asserted Identity	<sip_prop>.passertedid.displayname <sip_prop>.passertedid.uri	The unique identifier of the source sending the SIP message. This identifier is used for authentication purposes if your SIP configuration requires trusted connections. **Note:** If you define the **P-Asserted-Identity** parameter for the SIP connection through the EPM, Experience Portal ignores any attempt by an application to change this identity.
Subject	<sip_prop>.subject	A summary of the call.
Organization	<sip_prop>.organization	The name of the organization to which the SIP element issuing the request or response belongs.
Priority	<sip_prop>.priority	The priority of the request.

Creating a custom header

Procedure

In the`session.connection.protocol.sip.unknownhdr` session variable array, add a name and value pair.

Example

To create a custom header with the name as `new_header` and value as `mycustomheader`, add the following to the `session.connection.protocol.sip.unknownhdr` array:

```
AVAYA_SIPHEADER.session.connection.protocol.sip.unknownhdr[0].name = "new_header"
AVAYA_SIPHEADER.session.connection.protocol.sip.unknownhdr[0].value = "mycustomheader"
```

Sample VoiceXML page setting SIP headers in a VoiceXML application

The following VoiceXML page sets various SIP headers on a bridged transfer.

```
<?xml version="1.0" ?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml" xml:lang="en-us" >
<form id="form0">
  <property name="AVAYA_SIPHEADER.session.connection.protocol.sip.from.displayname"
   value="kong, king"/>
  <property name="AVAYA_SIPHEADER.session.connection.protocol.sip.to.displayname"
  value="godzilla"/>
  <property
```

```
name="AVAYA_SIPHEADER.session.connection.protocol.sip.passertedid.displayname"
  value="authority"/>
  <property name="AVAYA_SIPHEADER.session.connection.protocol.sip.passertedid.uri"
  value="sip:1234@123.321.123.321"/>
  <property name="AVAYA_SIPHEADER.session.connection.protocol.sip.unknownhdr[0].name"
  value="Random"/>
  <property name="AVAYA_SIPHEADER.session.connection.protocol.sip.unknownhdr[0].value"
  value="This is an unknown header"/>
  <transfer name="t1" type="bridge" dest="tel:1234"/>
</form>
</vxml>
```

SIP UPDATE method

The SIP UPDATE method, as per RFC 3311, allows you to update parameters of a session. While a call is in a queue, Experience Portal allows the SIP UPDATE method to update the following parameter of the call:

- User-to-User Interface data

You can send multiple UPDATE messages after the initial INVITE is established and before the final response to the INVITE.

Note:

Experience Portal supports SIP UPDATE method only if the Allow header indicates support.

Sample SIP UPDATE Method

The following is an example of the SIP UPDATE method:

```
Via: SIP/2.0/UDP pc33.<domain_name>.com;branch=<branch id>
;received=<ip_address>
To: <sip: email id>;tag=<tag number>
From: <name>
<sip:email id>;tag= <tag number>
 Call-ID: <call_id>
 CSeq: 63104 OPTIONS
 Contact: <sip: email id>
 Contact: <mailto: email id>
 Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, UPDATE
 Accept: application/sdp
 Accept-Encoding: gzip
 Accept-Language: en
 Supported: foo
 Content-Type: application/sdp
 Content-Length: 274
```

Chapter 6: The Application Logging web service

The Application Logging web service for third-party speech applications

Avaya Experience Portal includes an Application Logging web service that lets you save application and call flow data information from third-party speech applications into the `vpappLog` table of the Experience Portal database. Experience Portal can then include information from these third-party applications when it generates the Application Detail report and Application Summary report.

The Application Logging web service conforms to all W3C standards and can be accessed through any web service client using the Avaya-provided Web Services Description Language (WSDL) file.

Best practices

Experience Portal supports Axis 2.0 Application Logging web service.

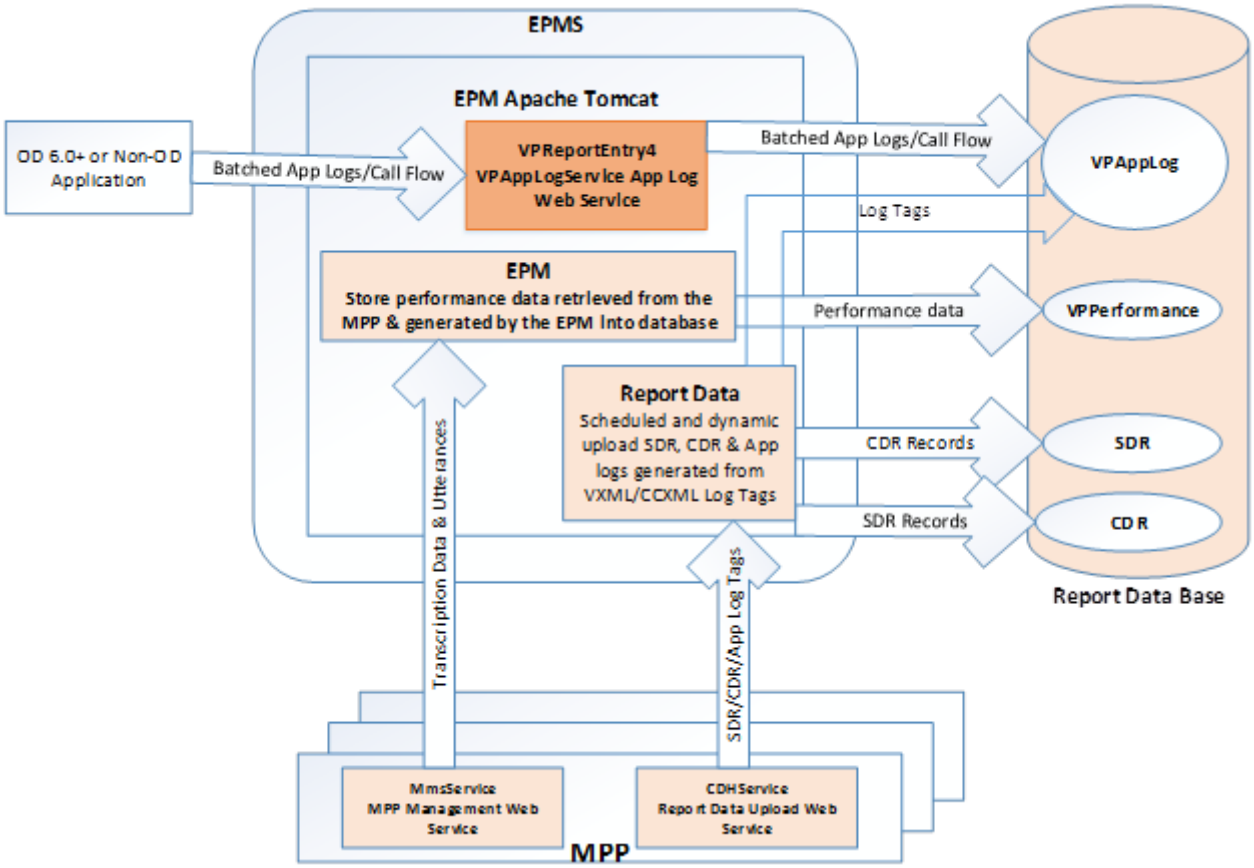
When using the Application Logging web service, keep in mind that:

- The Axis 2.0 Application Logging web service use Basic Authentication to authenticate web service client requests and only support HTTPS protocol.
- When the HTTPS protocol is used in the Web Service client, the Web Service client needs to handle to accept the certificate from the EPM server. No certificate needs to be installed on the Application Server. It is one-way SSL authentication.
- When calling the Axis 2.0 Application Logging web services, ensure to turn HTTP Chunking off.
- When you submit a request to the web service, you need to specify the user name and password which are specified in the Application Reporting section of the Web Service Authentication group on the EPM Settings page. If you need to change this user name or password, you must do it through the EPM. For details, see Configuring the Application Logging web service on page 63. For Axis 2.0 Application Logging web service requests, you can also use the Experience Portal web user name and password. The web user has to have the Web Services role with Application Reporting feature enabled.

- You must send all of the log entries for one or more session blocks at the same time. Do not send each log entry as it occurs or you may adversely affect Experience Portal system performance.
- You must use the logApplicationEventAlarm web service method with utmost caution. You should implement a throttling mechanism on the client side to limit flooding the EPM with the application events and alarms.
- You should have a proper queuing mechanism or sampling rate control method in place if you are sending a large number of log entries to the database. Otherwise this may adversely affect Experience Portal system performance.
- You should save all log entries in case the EPM is unavailable when the Application Logging web service is called. That way you can resend the log entries when the EPM becomes available.
- If your Experience Portal EPM software runs on a dedicated server machine, you should configure a auxiliary EPM server to handle Application Logging web service requests if the primary EPM server is unavailable.
- The Avaya Experience Portal Application Detail report and Application Summary report expect the report data to be in a particular format. For details about what Application Detail Records (ADRs) are stored in the `vpapplog` table, see Generating custom reports using third-party software in *Administering Avaya Experience Portal*.

Application Logging web service flow diagram

The following figure shows how speech applications interact with the Application Logging web service to add application messages to the Experience Portal database.



Configuring the Application Logging web service

About this task

To configure the Application Logging web service, you need to download the Avaya-provided WSDL file and build a custom web service client based on that file.

Procedure

1. Log on to the EPM web interface by using an account with the Administration user role.
2. From the EPM main menu, select **System Configuration** > **EPM Server**.
3. On the EPM Servers page, click **EPM Settings**.
4. On the EPM Settings page, go to the **Application Reporting** section in the **Web Service Authentication** group.
5. Enter the user name and password that must be included with all Application Logging web service requests for Digest Authentication.

This is the same user name and password that you should use when accessing the web service though the WSDL file.

6. Click **OK**.
7. For Axis 2.0 Application Logging web services: open the `https://<EPM-server>/axis2/services/VPAppLogService` page in a web browser.

   Where *<EPM-server>* is the domain name or IP address of the system on which the primary EPM software is installed.
8. When prompted, enter the user name and password specified in the **Application Reporting** section of the EPM Settings page.
9. Save the WSDL file and use it to build the web service client that accesses the Application Logging web service. This web service conforms to all current W3C standards.

Application Logging web service methods

The Application Logging web service includes the following methods:

- reportBatch method for application logging on page 65
- reportBatch method for call flow data on page 67
- logFailed method on page 64
- logApplicationEventAlarm method for application Logging / Alarming on page 70

logFailed method

The Application Logging web service `logFailed` method adds the event `PALOG00017` to the Experience Portal database.

Parameters

Parameter	Type	Description
lastVPMSDown	Long integer	The timestamp of the last time the EPM was down.

Return values

There are no return values from this method.

reportBatch method for application logging

The `reportBatch` method can be used to create application or call flow data log entries in the Experience Portal report database. In either case, the list of input parameters is the same. The only difference is the information you specify for each input parameter.

This topic discusses the information you should specify to create an application log entry. For information about creating a call flow data log entry, see reportBatch method for call flow data on page 67.

Parameters

Parameter	Type	Description
appServerAddress	string	The hostname or IP address of the application server.
applicationID	string	The name of the application. For example: `CollectTicketInfo` **Important:** The applicationID must match the name that was specified when the application was added to Avaya Experience Portal through the EPM. For an application that is assigned to a non-default zone, include the zone information using the format: `ZoneId:applicationName` You can view all application names on the Applications page. You can retrieve the zone Id information using the getZoneInfo method of the Management Interface web service.
level	string	The logging level. The options are: • `Fatal` • `Error` • `Warning` • `Info`
reason	string	The reason this log entry was made. For example: `Application ended successfully.` The reason for the first log entry in a session block should always be "–" (dash).
sessionID	string	The session ID for the session. This is a user-defined identifier that should be unique across sessions.

Table continues…

Parameter	Type	Description
timestamp	string	The current system time in milliseconds since January 1, 1970 00:00:00 UTC. The value of this parameter should contain a long number.
transactionName	string	The transaction name that this log entry is a part of. For example: `Hung Up` **Important:** Every transaction should start with a log entry setting the transaction name along with the activity type of `Start`. Once you start a transaction, all log entries should use the same transaction name up to and including the final log entry for that transaction.
type	string	The activity type for this log. The options are: • `Start` • `In Progress` • `End` • `Cancel`
userLog	string	The session label.
varName	string	A variable name, if one should be included with this log entry.
varValue	string	The value of the variable named in varName.
activityDuration	integer	The duration in seconds between the timestamp of the first log entry with the same transaction name and the activity type of `Start` and this log entry in a single session block. **Note:** This calculation should be based on one block of log entries that belong in one session.
moduleIdNodeId	string	The module ID and node ID in the format: `[Module Id]:Node Id`, where `Module Id` is only specified if it is *not* the same as the application name. For example, if the application name is `CollectTicketInfo` and it contains the `CollectTicketInfo` module with the node `StartTicket` and the `GetPayment` module with the node `StartPay`, you would specify them as: • `:StartTicket` • `GetPayment:StartPay`

Return values

The `reportBatch` method returns one of the following values:

- `success`
- `decryption failed - error occured decrypting the password`
- `Password incorrect`
- `Request is out of date`
- `Error storing data in database`
- `Error initializing database`
- `Error getting the EPID`

If the method fails, you can use the `logFailed` method to enter an event into the Experience Portal event log.

reportBatch method for call flow data

The `reportBatch` method can be used to create application or call flow data log entries in the Experience Portal report database. In either case, the list of input parameters is the same. The only difference is the information you specify for each input parameter.

This topic details the information you should specify to create a call flow data log entry. For information about creating an application log entry, see reportBatch method for application logging on page 65.

Parameters

Parameter	Type	Description
appServerAddress	string	The hostname or IP address of the application server.

Table continues…

Parameter	Type	Description
applicationID	string	The name of the application. For example: `CollectTicketInfo` **Important:** The applicationID must match the name that was specified when the application was added to Avaya Experience Portal through the EPM. For an application that is assigned to a non-default zone, include the zone information using the format: `ZoneId:applicationName` You can view all application names on the Applications page. You can retrieve the zone Id information using the getZoneInfo method of the Management Interface web service.
level	string	This should always be: `Info`
reason	string	The reason this log entry was made. When entering data in this field, you should keep in mind that: • The reason for the first log entry in a session block should always be "–" (dash). • If the activity type is `Node Entry` or `Application Exit`, this field must contain the moduleIdNodeId of the previous log entry of type `Node Entry`. This allows Experience Portal to track the source of the breadcrumb. • If the activity type is `Module Exit`, this field can contain whatever reason code you want to use.
sessionID	string	The session ID for the session. This is a user-defined identifier that should be unique across sessions.
Timestamp	string	The current system time in milliseconds since January 1, 1970 00:00:00 UTC. The value of this parameter should contain a long number.
transactionName	string	This should always be: `Framework`
type	string	The options are: • `Node Entry` • `Module Exit` • `Application Exit`
userLog	string	The session label.
varName	string	A variable name, if one should be included with this log entry.

Table continues…

Parameter	Type	Description
varValue	string	The value of the variable named in varName.
activityDuration	integer	The duration in seconds between the timestamp of the first log entry with the activity type of `Node Entry` or `Application Exit` and this log entry in a single session block. **Note:** This calculation should be based on one block of log entries that belong in one session.
moduleIdNodeId	string	The module and node identifiers. If the type of the last log entry in the session block is `Application Exit`, than this field should be "`--`" (dash dash). Otherwise, it should contain the module ID and node ID in the format: `[Module Id]:Node Id`, where `Module Id` is only specified if it is *not* the same as the application name. For example, if the application name is `CollectTicketInfo` and it contains the `CollectTicketInfo` module with the node `StartTicket` and the `GetPayment` module with the node `StartPay`, you would specify them as: • `:StartTicket` • `GetPayment:StartPay`

Return values

The `reportBatch` method returns one of the following values:

- `success`
- `decryption failed - error occured decrypting the password`
- `Password incorrect`
- `Request is out of date`
- `Error storing data in database`
- `Error initializing database`
- `Error getting the VPID`

If the method fails, you can use the `logFailed` method to enter an event into the Experience Portal event log.

logApplicationEventAlarm method for application Logging / Alarming

The logApplicationEventAlarm method can be used to issue the application events / alarms from the application to the EPM. The alarm events are sent via SNMP traps to the configured network management station but not through INADS.

This topic discusses the information you need to specify to create an application event entry.

Parameters

Parameter	Type	Description
appServerAddress	string	The hostname or IP address of the application server.
applicationID	string	The name of the application. For example: `CollectTicketInfo`
level	string	The logging level. The options are: • `Fatal` • `Error` • `Warning`
reason	string	The generated message defined by the application. Internationalization must be provided by the application. The maximum length is 100 characters. If the length is over the maximum, it will be truncated.
sessionID	string	The session ID for the session.
timestamp	string	The current system time in milliseconds since January 1, 1970 00:00:00 UTC. The value of this parameter should contain a long number.
transactionName	string	Not used.
type	string	Not used.
userLog	string	Not used.
varName	string	A variable name, if one should be included with this event entry. This is optional.
varValue	string	The value of the variable named in *varName*. This is optional.
activityDuration	integer	Not used.
moduleIdNodeId	string	Not used.

Return values

The logApplicationEventAlarm method returns one of the following values:

- `success`
- `decryption failed - error occured decrypting the password`
- `Password incorrect`

- `Request is out of date`
- `Error storing data in database`
- `Error initializing database`
- `Error getting the EPID`

The following table displays the log events and the associated messages to Alarm Maps:

Log Message	Message to Alarm Map	Log Level	Alarm Severity	Log Message Alarm Message
PAPP_00001	QAPP_00001	Warning	Minor	Application {0} reported an error from {1} at {2} with message: {3} {4} QAPP_00001: Application generated a Minor alarm
PAPP_00002	QAPP_00002	Error	Major	Application {0} reported an error from {1} at {2} with message: {3} {4} QAPP_00002: Application generated a Major alarm
PAPP_00003	QAPP_00003	Fatal	Critical	Application {0} reported an error from {1} at {2} with message: {3} {4} QAPP_00003: Application generated a Critical alarm

The parameter definitions for the log entries are:

- {0} – Name of the application defined on the application configuration web page *(applicationID)*
- {1} – Application server IP or host name *(appServerAddress)*, Session ID: *(sessionID)*
- {2} – Application server time of the log event *(timestamp)*
- {3} – The generated message defined by the application. Internationalization must be provided by the application *(reason)*
- {4} – Variable Name: *(varName)* Variable Value: *(varValue)*

Sample Application Logging web service WSDL file

The following is an example of the Application Logging web service WSDL file. The actual file is installed on the server that is running the EPM software. For details about accessing this file, see Configuring the Application Logging web service on page 63.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:com.avaya.vp.report.EPReport4"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
```

```
xmlns:impl="urn:com.avaya.vp.report.EPReport4"
xmlns:intf="urn:com.avaya.vp.report.EPReport4"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 2.0
Built on Apr 22, 2006 (06:55:48 PDT)-->
<wsdl:types>
 <schema elementFormDefault="qualified"
  targetNamespace="urn:com.avaya.vp.report.EPReport4"
  xmlns="http://www.w3.org/2001/XMLSchema">
 <complexType name="EPReportEntry4">
   <sequence>
    <element name="appServerAddress" nillable="true" type="xsd:string"/>
    <element name="applicationID" nillable="true" type="xsd:string"/>
    <element name="level" nillable="true" type="xsd:string"/>
    <element name="reason" nillable="true" type="xsd:string"/>
    <element name="sessionID" nillable="true" type="xsd:string"/>
    <element name="timestamp" nillable="true" type="xsd:string"/>
    <element name="transactionName" nillable="true" type="xsd:string"/>
    <element name="type" nillable="true" type="xsd:string"/>
    <element name="userLog" nillable="true" type="xsd:string"/>
    <element name="varName" nillable="true" type="xsd:string"/>
    <element name="varValue" nillable="true" type="xsd:string"/>
    <element name="activityDuration" type="xsd:int"/>
    <element name="moduleIdNodeId" nillable="true" type="xsd:string"/>
   </sequence>
 </complexType>
  <element name="reportBatch">
   <complexType>
    <sequence>
     <element maxOccurs="unbounded" name="entries" type="impl:EPReportEntry4"/>
     </sequence>
   </complexType>
  </element>
 <element name="reportBatchResponse"
 </complexType>
  </element>
  <element name="reportBatchReturn" type="xsd:string"/>
  </sequence>
 </complexType>
  </element>
  <element name="logFailed">
   <complexType>
    <sequence>
     <element name="lastVpmsDown" type="xsd:long"/>
    </sequence>
   </complexType>
  </element>
  <element name="logFailedResponse">
    </complexType>
  </element>
  <element name="logApplicationAlarm">
  </complexType>
  <sequence>
  <element maxOccurs="unbounded" name="entries" type="impl:EPReportEntry4"/>
  <sequence>
  </complexType>
  </element>
  <element name="logApplicationAlarmResponse">
  </complexType>
  <sequence>
  <element name="logApplicationEventAlarmReturn" type="xsd:string"/>
  <sequence>
  <complexType/>
```

```
  </element>
 </schema>
</wsdl:types>
<wsdl:message name="reportBatchRequest">
 <wsdl:part element="impl:reportBatch" name="parameters"/>
</wsdl:message>
<wsdl:message name="logFailedRequest">
 <wsdl:part element="impl:logFailed" name="parameters"/>
</wsdl:message>
<wsdl:message name="logApplicationEventAlarmRequest">
<wsdl:part element="impl:logApplicationEventAlarm" name="parameters"/>
</wsdl:message>
<wsdl:message name="reportBatchResponse">
 <wsdl:part element="impl:reportBatchResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="logFailedResponse">
 <wsdl:part element="impl:logFailedResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="llogApplicationEventAlarmResponse">
<wsdl:part element="impl:logApplicationEventAlarmResponse" name="parameters"/>
</wsdl:message>
<wsdl:portType name="EPReport4">
 <wsdl:operation name="reportBatch">
  <wsdl:input message="impl:reportBatchRequest" name="reportBatchRequest"/>
  <wsdl:output message="impl:reportBatchResponse" name="reportBatchResponse"/>
 </wsdl:operation>
 <wsdl:operation name="logFailed">
  <wsdl:input message="impl:logFailedRequest" name="logFailedRequest"/>
  <wsdl:output message="impl:logFailedResponse" name="logFailedResponse"/>
</wsdl:operation>
 <wsdl:operation name="logApplicationEventAlarm">
  <wsdl:input message="impl:logApplicationEventAlarmRequest"
name="logApplicationEventAlarmRequest"/>
  <wsdl:output message="impl:logApplicationEventAlarmResponse"
name="logApplicationEventAlarmResponse"/>
 </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="EPReport4SoapBinding" type="impl:EPReport4">
 <wsdlsoap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="reportBatch">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="reportBatchRequest">
   <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="reportBatchResponse">
   <wsdlsoap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="logFailed">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="logFailedRequest">
   <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="logFailedResponse">
   <wsdlsoap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="logApplicationEventAlarm">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="logApplicationEventAlarmRequest">
   <wsdlsoap:body use="literal"/>
  </wsdl:input>
<wsdl:output name="logApplicationEventAlarmResponse">
   <wsdlsoap:body use="literal"/>
```

```
  </wsdl:output>
 </wsdl:operation>
</wsdl:binding>
<wsdl:service name="EPReport4Service">
 <wsdl:port binding="impl:EPReport4SoapBinding" name="EPReport4">
  <wsdlsoap:address location="http://localhost:80/EPReport4"/>
 </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Chapter 7: The Application Interface web service

The Application Interface web service

Developers can use the Application Interface web service to:

- Start a CCXML or VoiceXML application that has been added to Avaya Experience Portal using the Add Application page.

  The web service automatically examines each MPP in the Experience Portal system and starts the session on the first available MPP that has the required outbound resources available.
- Send an event to a specific application session running on an MPP.
- Query the system for the total number of:
  - Used and unused outbound resources available
  - Unused SIP outbound resources
  - Unused H.323 outbound resources
- Send an SMS message or email message using one of the Avaya Experience Portal configured SMS or Email connections.
- Start an SMS or email application that has been added to Avaya Experience Portal using the Add Application page.

The Application Interface web service conforms to all W3C standards and can be accessed through any web service client using the Avaya-provided Web Services Description Language (WSDL) file.

Tip:

Sample files showing how you can communicate with the Application Interface web service using such methods as Java, JavaScript, and php are located in the `Support/Examples/Application Interface Web Service` directory on the Experience Portal installation DVD.

You can use the Application Interface test client to validate the Application Interface web service and the Experience Portal outcall, SMS, and Email functionality. Avaya supplies an installation script that automatically installs the Application Interface test client when Experience Portal is installed. For more information , see the *Configure and run the Application Interface test client*

section in the *Implementing Avaya Experience Portal on a single server* guide or the *Implementing Avaya Experience Portal on multiple servers* guide.

Best practices

When using the Application Interface web service, keep in mind that:

- The Axis 2.0 Application Interface web service uses Basic Authentication to authenticate web service client requests. When you submit a request to the web service, you need to include the user name and password that is configured as one of the Experience Portal users and that user must have the Web Services role checked.
- If non-ASCII characters are sent in the URL request to the web service they should be encoded as UTF-8 prior to sending the request.

  For example, an application name of ‘aña’ is encoded and sent as ‘a%C3%B1a’. Note that the non-ASCII character ‘ñ’ is sent as the UTF-8 value of ‘%C3%B1’.
- A non zero timeout value should be specified when using the `LaunchVXML` method. If no timeout value is passed, or if the value is 0, then a default value of 120 seconds will be used.
- For the methods `LaunchCCXML`, `LaunchVXML`, `LaunchSMS`, `LaunchEmail`, `SendSMS`, and `SendEmail`, parameters must be passed as name value pairs. For example, `parameter1=value`.

  Each name value pair is separated by semicolon. If a value of the specific parameter has multiple values, those values should be separated by comma (,). If the value contains a semicolon (;), you must escape it by using `\;`
- If you plan to use a single CCXML application to start multiple outgoing calls simultaneously, keep in mind that each application is handled by a single MPP, which means that each application is limited to the number of ports available on the MPP to which it is assigned. While the Application Interface web service tries to select the best MPP to handle the call, the application must have a way to verify the number of available ports so that it does not exceed the resources available on the MPP.

  If you want to make additional calls, you can either:

  - Start one additional instance of the CCXML call blast application for each additional MPP in the system.
  - Use the `QueryResources` method to check the available resources and start another instance of the CCXML call blast application as soon as enough resources are available.
- If your Experience Portal EPM software runs on a dedicated server machine, you should configure a auxiliary EPM server to handle Application Interface web service requests if the primary EPM server is unavailable.

Application Interface web service flow diagram

The following figure shows how an external application interacts with the Application Interface web service and how the Application Interface web service interacts with the rest of the Experience Portal system.

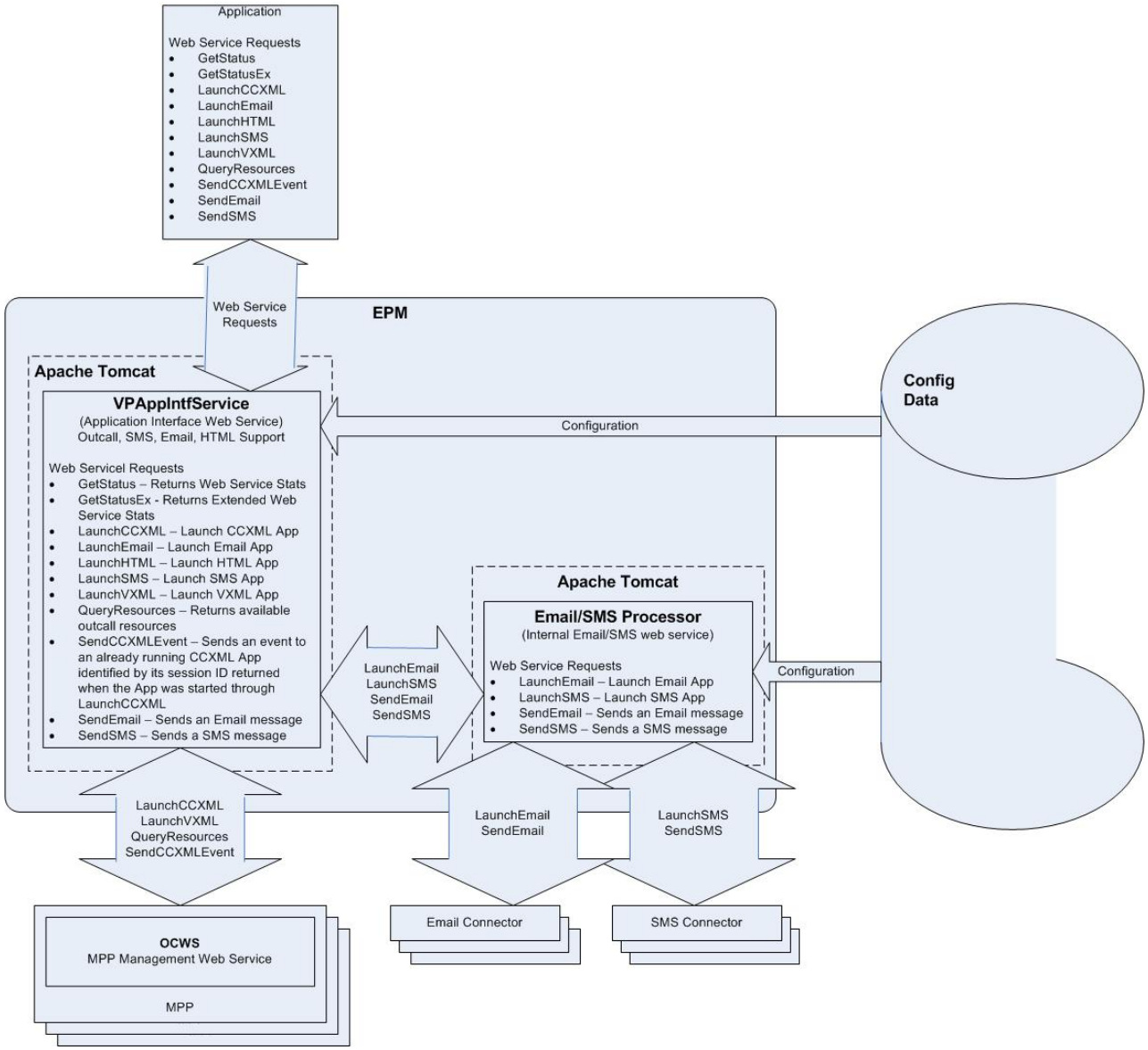


Figure 1: Application Interface Web service flow diagram

Configuring the Application Interface web service

Procedure

1. Log on to the EPM web interface by using an account with the Administration user role.
2. From the EPM main menu, select **User Management** > **Users**.
3. Select a user that will be used when sending requests to the Web Service.
4. On the Change User page, click the **Web Services** role to enable the user to use web service.
5. Click **Save**.
6. Open the following page in a Web browser: `https://<EPM-server>/axis2/services/VPAppIntfService?wsdl`

   Where *<EPM-server>* is the domain name or IP address of the system where the primary EPM software is installed.

   Note:

   When using a SOAP web service client, ensure that the URL used for invoking the web service is similar to `https://<EPM-server>/axis2/services/VPAppIntfService?wsdl`.
7. When prompted, enter the user name and password for the Experience Portal with the web services role checked.
8. Save the WSDL file and use it to build the web service client that accesses the Application Interface web service (axis 2.0). This web service conforms to all current W3C standards.

Application Interface web service methods

The Application Interface web service includes the following methods:

- SendSMS method on page 100

All methods can be called from any application that is running on the same network as theAvaya Experience Portal EPM server.

GetStatus method

This method returns the:

- Number of SIP requests processed since the Application Interface web service last started
- Number of telephony requests processed since the Application Interface web service last started
- Number of VoiceXML requests since the Application Interface web service last started
- Number of CCXML event requests since the Application Interface web service last started
- Number of Send CCXML requests sent since the Application Interface web service last started
- Maximum, minimum, and average number of MPP servers examined before a suitable MPP was found to run the requested application
- Date and time that the Application Interface web service was started
- Date and time on which the last request was made
- The version of the EPM software on the server running the Application Interface web service

Important:

The return values are described Return values on page 102.

Data Returned

Parameter	Type	Description
iSIPRequestsProcessed_returned	Integer	Total number of SIP requests processed since the Application Interface web service was started.
iTELRequestsProcessed_returned	Integer	Total number of H.323 or SIP telephony requests processed since the web service was started.
iVXMLRequestsProcessed_returned	Integer	Total number of VoiceXML requests processed since the web service was started.
iCCXMLRequestsProcessed_returned	Integer	Total number of CCXML requests processed since the web service was started.
iCCXMLEventsSent_returned	Integer	Total number of CCXML events sent since the web service was started.

Table continues…

Parameter	Type	Description
maxMPPHops_returned	Integer	Maximum number of MPPs that were examined before the web service found an MPP on which to run the requested application.
minMPPHops_returned	Integer	Minimum number of MPPs that were examined before the web service found an MPP on which to run the requested application.
avgMPPHops_returned	Integer	Average number of MPPs that were examined before the web service found an MPP on which to run the requested application.
lastRequestDateTime_returned	Date	Date and time on which the last request was made.
serviceStartedDateTime_returned	Date	Date and time on which the web service was last started.
vpmsSoftwareVersion_returned	String	The version of the EPM software running on the server.

GetStatusEx method

This method returns the:

- Number of SIP requests processed since the Application Interface web service last started
- Number of telephony requests processed since the Application Interface web service last started
- Number of VoiceXML requests since the Application Interface web service last started
- Number of CCXML requests since the Application Interface web service last started
- Number of Send CCXML event requests sent since the Application Interface web service last started
- Maximum, minimum, and average number of MPP servers examined before a suitable MPP was found to run the requested application
- Date and time that the Application Interface web service was started
- Date and time on which the last request was made
- The version of the EPM software on the server running the Application Interface web service
- Number of `LaunchEmail` requests since the Application Interface web service last started
- Number of `LaunchSMS` requests since the Application Interface web service last started
- Number of `SendEmail` requests since the Application Interface web service last started
- Number of `SendSMS` requests since the Application Interface web service last started

Important:

The return values are described Return values on page 102.

Data returned

Parameter	Type	Description
iSIPRequestsProcessed_returned	Integer	Total number of SIP requests processed since the Application Interface web service was started.
iTELRequestsProcessed_returned	Integer	Total number of H.323 or SIP telephony requests processed since the web service was started.
iVXMLRequestsProcessed_returned	Integer	Total number of VoiceXML requests processed since the web service was started.
iCCXMLRequestsProcessed_returned	Integer	Total number of CCXML requests processed since the web service was started.
iCCXMLEventsSent_returned	Integer	Total number of CCXML events sent since the web service was started.
maxMPPHops_returned	Integer	Maximum number of MPPs that were examined before the web service found an MPP on which to run the requested application.
minMPPHops_returned	Integer	Minimum number of MPPs that were examined before the web service found an MPP on which to run the requested application.
avgMPPHops_returned	Integer	Average number of MPPs that were examined before the web service found an MPP on which to run the requested application.
lastRequestDateTime_returned	Date	Date and time on which the last request was made.
serviceStartedDateTime_returned	Date	Date and time on which the web service was last started.
vpmsSoftwareVersion_returned	String	The version of the EPM software running on the server.
iLaunchAppRequestsProcessed_returned	Integer	Total number of `LaunchApp` requests processed since the Application Interface web service was started.
iLaunchEmailRequestsProcessed_returned	Integer	Total number of `LaunchEmail` requests processed since the Application Interface web service was started.
iLaunchSmsRequestsProcessed_returned	Integer	Total number of `LaunchSMS` requests processed since the Application Interface web service was started.
iSendAppRequestsProcessed_returned	Integer	Total number of `SendApp` requests processed since the Application Interface web service was started.
iSendEmailRequestsProcessed_returned	Integer	Total number of `SendEmail` requests processed since the Application Interface web service was started.
iSendSmsRequestsProcessed_returned	Integer	Total number of `SendSMS` requests processed since the Application Interface web service was started.

LaunchCCXML method

This method starts the specified CCXML application, and, if successful, returns the:

- Experience Portal session ID for the new session
- Total number of outbound resources available, both used and unused
- Total number of unused SIP outbound resources
- Total number of unused H.323 outbound resources

Important:

The CCXML application started *must* return the status of that start using a custom event as described in Returning the status of a LaunchCCXML request on page 85. For information on return values, see CCXML application status return values on page 84. The return values are described Return values on page 102.

Note:

If a CCXML application starts multiple VoiceXML applications, each of these applications can use a different speech server. Prior to Experience Portal 7.0, multiple applications used only one speech server. This feature comes into play if you have different languages loaded onto different speech servers.

When a CCXML application starts a VoiceXML dialog by name, the configured ASR or TTS is used. If the configured ASR or TTS changes, the current ASR or TTS resources are released and new resources are seized before the dialog starts.

For example, a CCXML application can start a dialog with English ASR or TTS. The application can determine that the caller prefers Spanish and so, the second dialog is opened with Spanish ASR or TTS.

Parameters

Parameter	Type	Description
toURI	String	Provides a hint to the Application Interface web service about what resources this CCXML application requires. The options are: • Blank (no input): Indicates that there are no requirements. • `tel`: Use this for an H.323 or SIP connection, or a mix of both. • `sip`: Use this for a standard SIP connection. • `sips`: Use this for a secure SIP connection.

Table continues…

Parameter	Type	Description
applicationName	String	Name of the CCXML application to run once the outbound call has connected. **Important:** The applicationName must match the name that was specified when the application was added to Avaya Experience Portal through the EPM. For an application that is assigned to a non-default zone, include the zone information using the format: `ZoneId:applicationName` You can view all application names on the Applications page. You can retrieve the zone Id information using the getZoneInfo method of the Management Interface web service.
applicationURL	String	Parameters that should be appended to URL specified for the application in the EPM. This allows you to invoke the application with different arguments as needed.
parameters	String	One or more name-value variable pairs that will be passed to the CCXML application when it is invoked. Each pair should be in the format `parametername=value`, and multiple pairs should be separated by a `;` (semi-colon). **Note:** When the web service passes the parameter to the application, it appends the namespace `session.values.avaya.ParameterMap`. Therefore, the variable should be referenced in your application as `session.values.avaya.ParameterMap.`*`parametername`*. For example, if you specify `UserCounter=0` in the web service, you would reference `session.values.avaya.ParameterMap.UserCounter` in your application.
uuiInfo	String	The Application Interface web service passes any information in this parameter to the platform telephony layer included in the outbound call.
launchTimeout	Integer	The maximum amount of time, in seconds, to wait for the CCXML application to be start before returning an error message.
zone	String	Zone name where request should be directed. This is the zone name. (Optional).

Data returned

Parameter	Type	Description
sessionID_returned	String	If the CCXML application connected successfully, the Application Interface web service sets this return value to the session ID Experience Portal assigned to the new CCXML session.
totalRes_returned	Integer	Total number of outbound resources available, both used and unused.
unusedSIP_returned	Integer	Total number of unused SIP outbound resources.
unusedH323_returned	Integer	Total number of unused H.323 outbound resources.

Note:

If zones are enabled, the returned resource information is the information for either the zone specified in the request, or for the default zone, if no zone was specified.

CCXML application status return values

Status	App Intf WS rc	Application
"success" `	0	
"networkdisconnect"	0	
"nearenddisconnect"	0	
"farenddisconnect"	0	
"calltransferred"	0	
"parse error"	0x22 (Invalid URL)	
"uri not found"	0x22 (Invalid URL)	
"fetch timeout"	0x13 (Failed)	LaunchCCXML only
"web server error"	0x13 (Failed)	LaunchCCXML only
"fetch error"	0x13 (Failed)	LaunchCCXML only
"unknown error"	0x13 (Failed) v	LaunchCCXML only
"noresource"	0x2 (No Resource)	
"busy"	0x10 (Busy)	
"networkbusy"	0x10 (Busy)	
"noanswer"	0x11 (No Answer)	
"noroute"	0x20 (Invalid URI)	LaunchVXML only
"unknown"	0x12 (Network Refusal)	LaunchVXML only
"internalerror"	0x12 (Network Refusal)	LaunchVXML only
"glare"	0x12 (Network Refusal)v	LaunchVXML only
"invalidstate"	0x12 (Network Refusal)	LaunchVXML only
"fax detected"	0x16 (Fax Detected)	

Note:

All return values generated by the LaunchVXML and LaunchCCXML may not have a mapping to the status that the CCXML application sends.

Returning the status of a LaunchCCXML request

About this task

If you invoke a CCXML application using the Application Interface web service `LaunchCCXML` method, the application started *must* return the status of that start using a custom event. This allows the CCXML application to determine whether the start was successful instead of relying on the limited information available to the Application Interface web service.

For example, if the CCXML application starts correctly but no one answers the outgoing call, the Application Interface web service still considers the call to be a success because the application started correctly. The CCXML application, on the other hand, may consider that call a failure because no one answered. In this case, the CCXML application would return a failure code to the Application Interface web service, and the Application Interface web service would return the failure code to Experience Portal.

Note:

For information on the status related return values that the CCXML application sends, see CCXML application status return values on page 84.

Procedure

1. Create a custom event handler in your application that sends the results back to the Application Interface web service.
2. In the custom event handler, create a variable called `status` that contains the status you want to return to the Application Interface web service.

   For example, if the call was not answered, you could assign `status` the value "no answer" using the `<var name="status" expr="no answer"/>` tag.
3. Send the response to the Application Interface web service using a `<send>` tag with the format: `<send name="avaya.launchresponse" targettype="avaya_platform" target="`*`session.id`*`" namelist="status"/>`, where *`session.id`* is the session identifier assigned to the session by Experience Portal.

CCXML session properties

session.values namespace properties

At the start of a CCXML session, the Application Interface web service places several properties in the `session.values.avaya` namespace properties.

Property	Description
`telephony.native_audio_format`	The audio encoding codec the MPP uses as the default for audio recording within the Avaya Voice Browser (AVB) when the speech application does not specify the format for recording caller inputs. The options are: • **audio/basic**: The AVB uses the mu-Law encoding format, which is used mostly in the United States and Japan. • **audio/x-alaw-basic**: The AVB uses the A-Law encoding format, which is used in most countries other than the United States and Japan.
`fax_detect_enabled`	For inbound calls, this property is set to the same value as the **Fax Detection Enable** parameter set for the application through the EPM web interface.
`fax_detect_redirect_uri`	For inbound calls, if fax detection is enabled, this property is set to the same value as the **Fax Phone Number** parameter set for the application through the EPM web interface.
`ConnectTimeoutSecs`	For outbound calls launched by the Application Interface web service, this is the maximum amount of time, in seconds, to wait for the CCXML application to be start before returning an error message.
`UUI_Info`	For outbound calls launched by the Application Interface web service, this property contains the information passed to the application in the uuiInfo parameter of the `LaunchCCXML` method.
`Parameters`	For outbound calls launched by the Application Interface web service, this property contains the name-value pairs passed to the application by the `LaunchCCXML` method.
`ParameterMap`	For outbound calls launched by the Application Interface web service, this object contains all of the parameters encoded in the `session.values.avaya` parameters field.

Additional properties

The CCXML browser also maintains current data in the `connections`, `conferences` and `dialogs` objects available within the `session` namespace. For details about these objects, see the W3C CCXML Version 1.0, W3C Working Draft dated 19 January 2007.

LaunchEmail method

This method starts the specified email type application and if successful, returns the Experience Portal session ID for the new session.

Note:

The launched applications is responsible for sending the email message using the information passed to the application.

Important:

For more details on the return values, see Return values on page 102.

Parameters

Parameter	Type	Description
to	String	Email address that is configured on the system and can be used by the application being launched.
from	String	Multiple values, using a comma (,) as a delimiter, that can be used by an application. For example: john@abc.com,sam@abc.com,bob@abc.com.
cc	String	Email CC. You can specify multiple values using a comma (,) as a delimiter. For example: john@abc.com,sam@abc.com,bob@abc.com. (Optional)
bcc	String	Email BCC. You can specify multiple values using a comma (,) as a delimiter. For example: john@abc.com,sam@abc.com,bob@abc.com. (Optional)
subject	String	The subject of the message. (Optional)
body	String	Body of the message. (Optional)
attachments	String	A list of attachment URLs, separated by a comma (,). (Optional)
headers	String	Contains name value in this format: name=value;name=value with a semicolon (;) as delimiter. Content-Type, Reply-To etc can be specified as headers. (optional)
applicationName	String	Name of the application to be launched. Contains zone information in the format zoneId:appName. **Important:** The applicationName must match the name that was specified when the application was added to Avaya Experience Portal through the EPM. For an application that is assigned to a non-default zone, include the zone information using the format: `ZoneId:applicationName` You can view all application names on the Applications page. You can retrieve the zone Id information using the getZoneInfo method of the Management Interface web service.

Table continues…

Parameter	Type	Description
appParameters	String	Parameters to be send to Orchestration Designer App. Contains name value in this format: name=value;name=; with ; as delimiter. (optional)
emailParameters	String	Parameters to be send to Orchestration Designer App. Contains name value in this format: name=value;name=; with ; as delimiter. See table below for parameters. (optional)
ucid	String	Unique customer ID. (Optional)
parentID	String	Caller's session ID. (Optional)
requestTimeout	Integer	Time to wait for the request to finish. (In seconds)

Data returned

Parameter	Type	Description
sessionID_returned	String	If the request completes successfully, this return value contains the session ID string

LaunchSMS method

This method starts the specified SMS type application and if successful, returns the Experience Portal session ID for the new session.

Note:

The launched applications is responsible for sending the SMS message using the information passed to the application.

Important:

The return values are described Return values on page 102.

Parameters

Parameter	Type	Description
to	String	Short code that is configured on the system and can be used by the application being launched.
from	String	Multiple values using a comma (,) as a delimiter, that can be used by an application. For example: 12345,23456,54321. (Mandatory)
message	String	Message to send. (Optional)

Table continues…

Parameter	Type	Description
applicationName	String	Name of the application to be launched. Contains zone information in the format: ZoneId:appName. **Important:** The applicationName must match the name that was specified when the application was added to Avaya Experience Portal through the EPM. For an application that is assigned to a non-default zone, include the zone information using the format: `ZoneId:applicationName` You can view all application names on the Applications page. You can retrieve the zone Id information using the getZoneInfo method of the Management Interface web service.
appParameters	String	Parameters to be sent to the Orchestration Designer app. Contains name value using semicolon (;) as delimiter, in the following format: name=value;name=;. (Optional)
smsParameters	String	Parameters to be sent to o SMSC. Contains name value using semicolon (;) as delimiter, in the following format: name=value;name=;. (Optional)
ucid	String	Unique caller ID. (Optional)
parentID	String	Caller's session ID. (Optional)
requestTimeout	Integer	Time to wait for the request to finish. (In seconds)

Data returned

Parameter	Type	Description
sessionID_returned	String	If the request completed successfully, this return value contains the session ID string.

LaunchHTML method

This method starts the specified HTML type application and if successful, returns the following:

- Experience Portal session ID for the new session
- The session ID created on the application server
- The application URI which the HTML application was launched

Important:

The return values are described in Return values on page 102.

Parameters

Parameter	Type	Description
applicationName	String	Name of the application to be launched. Contains zone information in the following format: ZoneId:appName. **Important:** The applicationName must match the name that was specified when the application was added to Avaya Experience Portal through the EPM. For an application that is assigned to a non-default zone, include the zone information using the format: `ZoneId:applicationName` You can view all application names on the Applications page. You can retrieve the zone Id information using the getZoneInfo method of the Management Interface web service.
appParameters	String	Parameters to be sent to the launched application. Contains name value using semicolon (;) as delimiter, in the following format: name=value;name=;. (Optional) If a value contains a semicolon (“;”) it needs to be escaped using backslash (“\”).
ucid	String	Unique caller ID. (Optional)
parentID	String	Caller’s session ID. (Optional)
requestTimeout	Integer	Time to wait for the request to finish. (In seconds) This value must be in the range 1-60 (inclusive).

Data returned

Parameter	Type	Description
sessionID_htlpApp	String	If the request completed successfully, this return value contains the Experience Portal session ID that was created when the application was launched.
sessionID_AppServer	String	If the request completed successfully, this return value contains the Session ID that was created on the application server.
applicationURL	String	If the request completed successfully, this return value contains the application URI against which the HTML application was launched.

LaunchVXML method

This method initiates an outbound call on an available MPP, then starts the specified VoiceXML application. If successful, it returns the:

- Experience Portal session ID for the new session
- Total number of outbound resources available, both used and unused
- Total number of unused SIP outbound resources
- Total number of unused H.323 outbound resources

The actual launch of the VoiceXML application is handled by the default CCXML page, which is also responsible for returning success or failure back to the Application Interface web service.

When the VoiceXML application is invoked, the first VoiceXML page is prepared. If this succeeds, the outbound call is placed by the system. If the call connects and the dialog starts without error, then the VoiceXML application returns a successful launch code. Otherwise, the application returns an appropriate error code.

Note:

If the initial VoiceXML page cannot be prepared for any reason, the Application Interface web service does not place the outbound call. Therefore a customer will never be bothered by an outbound call that cannot possibly start correctly.

For information about the status codes returned by the CCXML page, see Returning the status of a LaunchCCXML request on page 85.

Note:

If a CCXML application launches multiple VoiceXML applications, each of these applications can use a different speech server. Prior to Experience Portal 7.0, multiple applications used only one speech server. This feature comes into play if you have different languages loaded onto different speech servers.

When a CCXML application launches a VoiceXML dialog by name, the configured ASR or TTS is used. If the configured ASR or TTS changes, the current ASR or TTS resources are released and new resources are seized before the dialog starts.

For example, a CCXML application can launch a dialog with English ASR or TTS. The application can determine that the caller prefers Spanish and so, the second dialog is launched with Spanish ASR or TTS.

Important:

The return values are described Return values on page 102.

Parameters

Parameter	Type	Description
toURI	String	The number or destination to be contacted by the outbound application. This parameter can be prefixed with one of the following strings: • `tel`: Use this for an H.323 or SIP connection, or a mix of both. • `sip`: Use this for a standard SIP connection. • `sips`: Use this for a secure SIP connection.
fromURI	String	Calling address information to pass with the outbound call.
applicationName	String	Name of the VoiceXML application to run once the outbound call has connected. **Important:** The applicationName must match the name that was specified when the application was added to Avaya Experience Portal through the EPM. For an application that is assigned to a non-default zone, include the zone information using the format: `ZoneId:applicationName` You can view all application names on the Applications page. You can retrieve the zone Id information using the getZoneInfo method of the Management Interface web service.
applicationURL	String	Parameters that should be appended to URL specified for the application in the EPM. This allows you to invoke the application with different arguments as needed.

Table continues…

Parameter	Type	Description
parameters	String	One or more name-value variable pairs that will be passed to the VoiceXML application when it is invoked. Each pair should be in the format `parametername=value`, and multiple pairs should be separated by a `;` (semi-colon). **Note:** When the web service passes the parameter to the application, it appends the namespace `session.avaya.telephone`. Therefore, the variable should be referenced in your application as `session.avaya.telephone.`*`parametername`*. For example, if you specify `UserCounter=0` in the web service, you would reference `session.avaya.telephone.UserCounter` in your application. **Tip:** If you want to enable call classification for this call, see Call classification with the LaunchVXML method on page 48.
uuiInfo	String	The Application Interface web service passes any information in this parameter to the platform telephony layer included in the outbound call.
connectTimeoutSecs	Integer	The maximum amount of time, in seconds, to wait for the outbound call to be connected. Enter a value between 0 and 59. If this parameter is set to 0 (zero) or omitted, Experience Portal uses the default value of 120 seconds.
zone	String	This is the zone name where request should be directed. (Optional).

Data returned

Parameter	Type	Description
sessionID_returned	String	If the VoiceXML application connected successfully, the Application Interface web service sets this return value to the session ID Experience Portal assigned to the new CCXML session.
totalRes_returned	Integer	Total number of outbound resources available, both used and unused.
unusedSIP_returned	Integer	Total number of unused SIP outbound resources.
unusedH323_returned	Integer	Total number of unused H.323 outbound resources.

Note:

If zones are enabled, the resource information returned is the information for the zone specified in the request. If no zone is specified, the resource information returned is the information for the default zone.

Call classification with the LaunchVXML method

Call classification allows the VoiceXML application to return the appropriate status code based on whether a human, an answering machine, or a fax machine answers an outbound call.

Call classification parameters for the LaunchVXML method

The following call classification name-value pairs can be passed as parameters with the LaunchVXML method. For both parameters the default is `false`, which means that you must specify the name-value pair in order to enable the associated functionality.

Name-value pair	Description
`enable_call_classification=true`	This required parameter enables call classification.
`detect_greeting_end=true`	This optional parameter instructs the VoiceXML application to identify the end of a recorded greeting if an answering machine answers the outbound call.
`call_classification_recorded_msg_timeout = in mili secs (e.g. 30000 is 30 sec)`	This Optional parameter is to set wait timeout for “end of recorded greeting”, if an answering machine answers the outbound call. Default is 30 sec.
`call_classification_connectWhen = ( OnConnect or OnProgress)`	This optional parameter is to start the CPA engine (call classification) on either OnConnect or OnProgress. By default it is set to OnProgress. In case the engine is started before connect, early media will also be captured for call classification.
`call_classification_timeout=value`	Timeout for outbound call classification from engine. This optional parameter indicates how long the call classification function will run if it is unable to determine the classification. The value specified should be in milliseconds. If the value is not provided, the default timeout is 20 sec.

Call classifications

If you enable call classification, the VoiceXML application sends one of the following classifications to the application server using the query arguments on the URL:

Classification	Description
`live_voice`	If a human being answers the call, the application starts the previously-prepared VoiceXML dialog. **Note:** This is the default classification assigned to the VoiceXML session before the call is placed. If a human being does not answer the call, this classification must be changed.

Table continues…

Classification	Description
recorded_msg	If the LaunchVXML method was invoked with detect_greeting_end=false or if the detect_greeting_end parameter was not specified and an answering machine answers the call, the application terminates the previously-prepared VoiceXML dialog and starts a new dialog by sending the classification recorded_msg to the application server.
msg_end	If the LaunchVXML method was invoked with detect_greeting_end=true and an answering machine answers the call, the application terminates the previously-prepared VoiceXML dialog and starts a new dialog by sending the classification msg_end to the application server.
fax_answer_tone	If a fax machine answers the call, the application terminates and returns the error code fax detected (8206) to the Application Interface web service.
timeout	If the VoiceXML application does not send a classification change message to the CCXML page within a given period of time, the CCXML applications assumes that a live person has answered the phone and it starts the previously-prepared VoiceXML dialog.
*	All other classifications result in the status code for no answer () being returned the Application Interface web service.

VoiceXML session properties

At the start of a VoiceXML session, the Application Interface web service places several properties in the session.connection, session.avaya.telephone, and session.telephone namespaces.

session.connection namespace properties

Property	Description
call_tag	The unique identifier for the session assigned by the Media Server.
ccxml.namelist.*	If the CCXML application passes data to the VoiceXML dialog at the beginning of the session, this array variable contains a list of the variable names passed by the CCXML application.
ccxml.values.*	If the CCXML application passes data to the VoiceXML dialog at the beginning of the session, this array variable contains a list of the values for the variable names contained in ccxml.namelist.*.
local.uri	The Dialed Number Identification Service (DNIS) associated with the call that triggered the VoiceXML session.
protocol.name	The telephony protocol name. The options are: • h323 • sip
remote.uri	The Automatic Number Identification (ANI) associated with the call that triggered the VoiceXML session.

session.avaya namespace properties

Note:

For convenience, several of the `session.avaya` namespace properties are the same as the `session.connection` namespace properties.

Property	Description
`telephone.ani`	The Automatic Number Identification (ANI) associated with the call that triggered the VoiceXML session.
`telephone.call_tag`	The unique identifier for the session assigned by the Media Server.
`telephone.called_extension`	For calls using an H.323 connection, this is the telephony port servicing the VoiceXML session.
`telephone.callid`	The unique identifier for the call assigned by the Media Server.
`telephone.channel`	This property is reserved for future use.
`telephone.dnis`	The Dialed Number Identification Service (DNIS) associated with the call that triggered the VoiceXML session.
`telephone.startPage`	The full URL, including any query string parameters, used to fetch the first page of the VoiceXML session.
`uui.mode`	The User-to-User Interface (UUI) mode under which this application is operating. The options are: • `shared` • `service provider`
`uui.shared[]`	If `uui.mode` is `shared`, this property contains an array of the shared UUI data pieces in name-value format.

session.telephone namespace properties

Note:

For convenience, most of the `session.telephone` namespace properties are the same as the `session.connection` and `session.avaya` namespace properties.

Property	Description
`ani`	The Automatic Number Identification (ANI) associated with the call that triggered the VoiceXML session.
`dnis`	The Dialed Number Identification Service (DNIS) associated with the call that triggered the VoiceXML session.
`call_tag`	The unique identifier for the session assigned by the Media Server.
`called_extension`	For calls using an H.323 connection, this is the telephony port servicing the VoiceXML session.
`callid`	The unique identifier for the call assigned by the Media Server.
`channel`	This property is reserved for future use.

Table continues…

Property	Description
`startPage`	The full URL, including any query string parameters, used to fetch the first page of the VoiceXML session.
`*`	This property contains any parameters passed to the Application Interface web service through the `LaunchVXML` method.

QueryResources method

This method takes a snapshot of the current outbound usage across all MPPs in the Experience Portal system and returns:

- Total number of outbound resources available, both used and unused
- Total number of unused SIP outbound resources
- Total number of unused H.323 outbound resources

  If zones are enabled, the returned resource information is the information for all MPPs with the zone specified in the request, or for the default zone if no zone was specified.

You can use this method to determine the approximate availability of outbound resources before you use the `LaunchCCXML` or `LaunchVXML` method to start a new outbound session.

Keep in mind, however, that system usage is extremely dynamic. The `QueryResources` method only returns a snapshot of the current usage. It does not look for upcoming outbound calls or try to determine whether another LaunchCCXML or LaunchVXML command has just started and is about to claim one or more outbound resources.

In addition, this method reports the total number of outbound resources available across all MPPs in the Experience Portal system, or all MPPs in the specified zone, if zones are enabled. Each application only has access to the available ports on the MPP to which it is assigned. If your site has multiple MPPs, that means any single application will probably not have access to the total number of resources returned by this method. For more information on using applications that launch multiple outgoing calls, see Best practices on page 76.

Important:

The return values are described Return values on page 102.

Data Returned

Name	Type	Description
totalRes_returned	Integer	Total number of outbound resources available, both used and unused.
unusedSIP_returned	Integer	Total number of unused SIP outbound resources.
unusedH323_returned	Integer	Total number of unused H.323 outbound resources.

Note:

If zones are enabled, the resource information returned is the information for the zone specified in the request. If no zone is specified, the resource information returned is the information for the default zone.

SendCCXMLEvent method

This method instructs the MPP to dispatch a user-named event with an accompanying parameter string to the specified CCXML session.

Important:

The return values are described Return values on page 102.

Parameters

Name	Type	Description
sessionID	String	The session ID of an existing CCXML session. This parameter must match exactly the session ID assigned by Avaya Experience Portal when the session started.
eventName	String	The user-defined event that the MPP should send to the session.
parameters	String	Any user-defined parameters that the MPP should pass along with the event to the CCXML session.

SendEmail method

This method starts the specified email type application and if successful, returns the Experience Portal session ID for the new session.

Important:

The return values are described Return values on page 102.

Parameters

Parameter	Type	Description
from	String	Email address used to send the message
to	String	Target of the message. You can specify multiple values, using a comma (,) as a delimiter. For example: john@abc.com,sam@abc.com,bob@abc.com.

Table continues…

Parameter	Type	Description
cc	String	Email CC. You can specify multiple values, using a comma (,) as a delimiter. For example: john@abc.com,sam@abc.com,bob@abc.com. (Optional)
bcc	String	Email BCC. You can specify multiple values, using a comma (,) as a delimiter. For example: john@abc.com,sam@abc.com,bob@abc.com. (Optional)
subject	String	The subject of the message. (Optional)
body	String	Body of the message. (Optional)
attachments	String	A list of attachment URLs, separated by a comma (,). (Optional)
headers	String	Contains name value in this format: name=value;name=value with a semicolon (;) as delimiter. Content-Type, Reply-To etc can be specified as headers. (optional)
applicationName	String	Name of the application to be launched. Contains zone info in the format zoneId:appName.
emailParameters	String	Parameters to be send to Orchestration Designer App. Contains name value in this format: name=value;name=; with ; as delimiter. See table below for parameters. (optional)
ucid	String	Unique customer ID. (Optional)
sessionID	String	Caller's session ID.
requestTimeout	String	Time to wait for the request to finish. (In seconds)

Data returned

Parameter	Type	Description
messageID_returned	String	If the request completes successfully, this return value contains the session ID string

SendSMS method

This method sends an SMS message, and if successful, returns the Experience Portal message ID for the new session.

Important:

The return values are described Return values on page 102.

Parameters

Parameter	Type	Description
from	String	Short code used to send the message.
to	String	Target of the message. You can specify multiple values using a comma (,) as a delimiter. For example: 12345,23456,54321.
message	String	Message to send.
applicationName	String	name of the application to be launched. Contains zone information in the format: ZoneId:appName.
smsParameters	String	Parameters to be sent to o SMSC. Contains name value using semicolon (;) as delimiter, in the following format: name=value;name=;. (Optional)
ucid	String	Unique caller ID. (Optional)
sessionID	String	Caller's session ID. (Optional)
requestTimeout	Integer	Time to wait for the request to finish. (In seconds)

Data returned

Parameter	Type	Description
sessionID_returned	String	If the request completed successfully, this return value contains the session ID.

Additional parameters in the LaunchEmail and SendEmail methods

Parameter	Description
CC	The CC email address. You can separate multiple email addresses by comma (,).
BCC	The BCC email address. You can separate multiple email addresses by comma (,).
ReplyTo	The ReplyTo address to be set in the email.
Attachment	The attachment list. You can separate multiple attachment files by comma (,).
RegisteredDelivery	Demanding a registered delivery for the request. Value should be one of the SMTP notification value: -1 (Notify Never), 1 (Notify Success), 2 (Notify Failure), 4 (Notify Delay)
Priority	Assigns a priority level, in the range of to >=1 and <=5, to the message. The "X-Priority" value in the email is set accordingly so that email importance is honored by the SMTP servers. 1 indicates high priority and 5 as low. If none specified, set as 0 – no priority marking for the email.
ContentType	Email content type: text/plain or text/html.
Charset	The charset of the email.
DisplayName	Display name to be used for the email from address.

Additional parameters in the LaunchSMS and SendSMS methods

Parameter	Description
RegisteredDelivery	Demanding a registered delivery for the request. Accepted values: 0 and 1. 1= Requires a receipt; 0 (default) = No delivery receipt.
DataCoding	Defines the encoding scheme of the short message user data. Default is 0 to use the SMSC default Alphabet. Refer to the SMPP specifications for details (Short Message Peer to Peer Protocol Specification v3.4; section: 5.2.19).

Table continues…

Parameter	Description
Priority	Designates a priority level to the message. 4 priority levels are supported (0 being lowest and 3 being highest). Default to 0. Refer to the SMPP specification document section 5.2.14 for more details.
ValidityPeriod	Sets the validity period for this message, so that SMSC can ignore the message if it can't deliver within this time. Default to 0 for SMSC default validity period. Refer to the SMPP specification document section 5.2.16 & 7.1.1.
ScheduleDelivery	Indicates that SMSC need to schedule the message for delivery at the time specified. Default to 0 for immediate delivery.
Encoding	Defines the character encoding scheme of the message user data. Default is to use ASCII. For the list of character encoding, refer: `http://docs.oracle.com/javase/1.3/docs/guide/intl/encoding.doc.html`
Language	The SMPP language indicator of the short message (CMT-136 standard). Refer to the SMPP specification for details: (Short Message Peer to Peer Protocol Specification v3.4; section: 5.3.2.19).
UserMessageReference	A custom reference id that can be submitted along with the message that could be used by the application to build context and further interpretation. Optional parameter; not all SMSC vendor required to support.

Return Values

The following table describes the return values that the application interface web service receives when the system process the web services request:

Return value		Attributes	Condition	Get Status	Get StatusEx	Launch CCXML	Launch Email	Launch SMS	Launch VXML	Query Resources	Send CCXML Event	Send Email	Send SMS	LaunchHTML
hex														
0x0	0	Success	The web service successfully completed the request.	X	X	X	X	X	X	X	X	X	X	X
0x1	1	Failure	The web service was unable to complete the request. Verify that the EPM and MPP servers are communicating properly and retry the request.	X	X	X	X	X	X	X	X	X	X	
0x2	2	No resource	No resources are available to process the request.			X	X	X	X			X	X	
0x3	3	No User Access	The user is not allowed to access the web service.			X	X	X	X		X	X	X	X
0x4	4	No App Access	The user is not allowed to access the specified application.			X	X	X	X			X	X	X
0x10	16	Busy	The destination named in the toURI parameter is busy and cannot be reached.						X					
0x11	17	No Answer	The destination named in the toURI parameter did not answer within the amount of time specified in the connectTimeoutSecs parameter.						X					
0x12	18	Network Refusal	The outbound call was rejected due to a network error.						X					
0x13	19	Failed	The application was not started due to a problem with the application server.			X								
0x14	20	Timeout	The operation did not get a response within the amount of time specified in the timeout parameter.			X	X	X	X			X	X	
0x15	21	No Response	The session ended and the VoiceXML/CCXML page did not return a response to Experience Portal. This can happen if the VoiceXML/CCXML page is not designed to send a response, or if the page has an error and exits before the code that sends the response executes.			X			X					
0x16	22	Fax Detected	The outbound number is associated with a fax machine.			X			X					
0x20	32	Invalid URI	The toURI parameter contains an invalid URI specification.			X			X					
0x21	33	Unknown Application	The applicationName parameter does not match one of the application names configured though the EPM. You can view all application names on the Applications page.			X	X	X	X			X	X	X
0x22	34	Invalid URL	The applicationURL parameter contains an invalid URL specification.			X			X					

Return value		Attributes	Condition	Get Status	Get StatusEx	Launch CCXML	Launch Email	Launch SMS	Launch VXML	Query Resources	Send CCXML Event	Send Email	Send SMS	LaunchHTML
0x23	35	Invalid Session ID	No current session ID matched the one specified in the sessionID parameter. Make sure that the session ID is correct and that the session is still running on the MPP.								X	X	X	
0x24	36	Invalid Connect Timeout	The Connect timeout in LaunchVXML request (connectTimeoutSecs) is too long. The value specified must be less than the MaxLaunchVXMLConnectTimeout parameter in the voiceportal.properties file on Experience Portal.						X					
0x25	37	Invalid Zone	Invalid Zone specified.			X	X	X	X	X		X	X	X
0x30	48	MPP Down	The MPP service is not running and cannot service the request.								X			
0x31	49	MPP Stopped	The MPP service is running, but the MPP is not currently processing calls.								X			
0x32	50	MPP WS Failure	The required out call web service is not running on any MPP in the system. Therefore the call cannot be connected. Messages from the Application Interface web service appear in the OCWSServer.log file on the MPP, which is accessible from the Media Server Service Menu Log Directories page.			X	X	X	X		X	X	X	
0x43	67	Invalid Timeout	Invalid timeout specified				X	X				X	X	X
0x44	68	Invalid To	Invalid data in the "to" field									X	X	
0x45	69	Invalid From	Invalid data in the "from" field				X						X	
0x46	70	Invalid Message	Invalid data in the message field										X	
0x47	71	Invalid Request Data	Invalid data in the request				X	X				X	X	X
0x57	87	Exception from HTMLLauncher	The HTML Launcher generated an exception											X
0x58	88	HTMLLauncher Failure	The HTML Launcher generated a failure											X
0x59	89	HTMLLauncher returned unknown application	The HTML Launcher could not locate the specified application											X
0x5A	90	HTMLLauncher returned unspecified error	The HTML Launcher returned an error that is unspecified.											X

Return value		Attributes	Condition	Get Status	Get StatusEx	Launch CCXML	Launch Email	Launch SMS	Launch VXML	Query Resources	Send CCXML Event	Send Email	Send SMS	LaunchHTML
0x5B	91	HTML Launcher returned Not Licensed	The HTML Launcher was unable to launch the application due to licensing.											X
0x102	258	Fatal Error	The web service is in an error state that requires a restart of the EPM service.	X	X	X	X	X	X	X	X	X	X	
0x103	259	Not Initialized	The web service is not initialized and unable to accept requests.	X	X	X	X	X	X	X	X	X	X	
0x104	260	Shutting Down	The web service is shutting down and no new requests are being accepted.	X	X	X	X	X	X	X	X	X	X	
0x201	513	Network Failed before Connection Completed	Represents that the network failed before the connection was completed.			X			X					
0x202	514	Near End Disconnect before Connection Completed	Represents that the caller disconnected the call before the called number was connected.			X			X					
0x203	515	Near End Disconnect before Connection Completed	Far End Disconnected before the connection was completed.			X			X					
0x204	516	Transferred before Connection Completed	Represents that the call is transferred before the connection was completed.			X			X					
0x205	517	Unknown Failure before Connection Completed	Represents that the call cannot be completed due to unknown reason.			X			X					
0x206	518	Call Disconnected before Application Started	Represents that the call is disconnected before the application was started.						X					

Sample Application Interface web service WSDL file

The following is an example of the Application Interface web service WSDL file. The actual file is installed on the server that is running the EPM software. For details about accessing this file, see Configuring the Application Interface web service on page 78.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns1="http://org.apache.axis2/xsd"
xmlns:ns="http://xml.avaya.com/ws/VPAppIntf/VoicePortal"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:ax25="http://services.vp.avaya.com/xsd"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:ax21="http://rmi.java/xsd"
xmlns:ax22="http://io.java/xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
targetNamespace="http://xml.avaya.com/ws/VPAppIntf/VoicePortal">
    <wsdl:types>
        <xs:schema xmlns:ax23="http://io.java/xsd" attributeFormDefault="qualified"
elementFormDefault="qualified" targetNamespace="http://rmi.java/xsd">
            <xs:import namespace="http://io.java/xsd"/>
            <xs:complexType name="RemoteException">
                <xs:complexContent>
                    <xs:extension base="ax23:IOException">
                        <xs:sequence>
                            <xs:element minOccurs="0" name="cause" nillable="true"
type="xs:anyType"/>
                            <xs:element minOccurs="0" name="message" nillable="true"
type="xs:string"/>
                        </xs:sequence>
                    </xs:extension>
                </xs:complexContent>
            </xs:complexType>
        </xs:schema>
        <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://services.vp.avaya.com/xsd">
```

```
            <xs:complexType name="SendCCXMLEventRequest">
                <xs:sequence>
                    <xs:element minOccurs="1" name="eventName" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="parameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="sessionID" nillable="false"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="SendCCXMLEventResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="retCode" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="QueryResourcesRequest">
                <xs:sequence>
                    <xs:element minOccurs="0" name="queryResourcesRequestUnused"
type="xs:int"/>
                    <xs:element minOccurs="0" name="zone" nillable="true"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="QueryResourcesResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="totalRes_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="unusedH323_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="unusedSIP_returned" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchVXMLRequest">
                <xs:sequence>
                    <xs:element minOccurs="1" name="applicationName" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="applicationURL" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="connectTimeoutSecs" type="xs:int"/>
                    <xs:element minOccurs="1" name="fromURI" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="parameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="toURI" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="uuiInfo" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="zone" nillable="true"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchVXMLResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="sessionID_returned"
nillable="false" type="xs:string"/>
                    <xs:element minOccurs="1" name="totalRes_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="unusedH323_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="unusedSIP_returned" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchCCXMLRequest">
                <xs:sequence>
                    <xs:element minOccurs="1" name="applicationName" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="applicationURL" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="launchTimeout" type="xs:int"/>
```

```
                    <xs:element minOccurs="0" name="parameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="toURI" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="uuiInfo" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="zone" nillable="true"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchCCXMLResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="sessionID_returned"
nillable="false" type="xs:string"/>
                    <xs:element minOccurs="1" name="totalRes_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="unusedH323_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="unusedSIP_returned" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchSMSRequest">
                <xs:sequence>
                    <xs:element minOccurs="0" name="appParameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="applicationName" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="from" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="message" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="parentID" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="requestTimeout" type="xs:int"/>
                    <xs:element minOccurs="0" name="smsParameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="to" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="ucid" nillable="true"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchSMSResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="sessionID_returned"
nillable="false" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="SendSMSRequest">
                <xs:sequence>
                    <xs:element minOccurs="1" name="applicationName" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="from" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="message" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="requestTimeout" type="xs:int"/>
                    <xs:element minOccurs="1" name="sessionID" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="smsParameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="to" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="ucid" nillable="true"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
```

```
            <xs:complexType name="SendSMSResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="messageID_returned"
nillable="false" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchEmailRequest">
                <xs:sequence>
                    <xs:element minOccurs="0" name="appParameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="applicationName" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="attachments" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="bcc" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="body" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="cc" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="emailParameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="from" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="headers" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="parentID" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="requestTimeout" type="xs:int"/>
                    <xs:element minOccurs="0" name="subject" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="to" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="ucid" nillable="true"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchEmailResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="sessionID_returned"
nillable="false" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="SendEmailRequest">
                <xs:sequence>
                    <xs:element minOccurs="1" name="applicationName" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="attachments" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="bcc" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="body" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="cc" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="emailParameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="from" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="headers" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="requestTimeout" type="xs:int"/>
                    <xs:element minOccurs="1" name="sessionID" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="subject" nillable="false"
```

```
type="xs:string"/>
                    <xs:element minOccurs="1" name="to" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="ucid" nillable="true"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="SendEmailResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="messageID_returned"
nillable="false" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
             <xs:complexType name="LaunchHtmlRequest">
                <xs:sequence>
                    <xs:element minOccurs="0" name="appParameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="applicationName" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="parentID" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="requestTimeout" type="xs:int"/>
                    <xs:element minOccurs="0" name="ucid" nillable="true"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchHtmlResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="applicationURL" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="sessionID_AppServer"
nillable="false" type="xs:string"/>
                    <xs:element minOccurs="1" name="sessionID_htmlApp" nillable="false"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchAppRequest">
                <xs:sequence>
                    <xs:element minOccurs="1" name="applicationName" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="from" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="genericAppType" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="parameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="0" name="parentID" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="requestTimeout" type="xs:int"/>
                    <xs:element minOccurs="1" name="to" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="ucid" nillable="true"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="LaunchAppResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="ID_returned" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="parameters" nillable="false"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="GetStatusExRequest">
                <xs:sequence>
```

```
                    <xs:element minOccurs="0" name="getStatusExRequestUnused"
type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="GetStatusExResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="avgMPPHops_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="iCCXMLEventsSent_returned"
type="xs:int"/>
                    <xs:element minOccurs="1" name="iCCXMLRequestsProcessed_returned"
type="xs:int"/>
                    <xs:element minOccurs="1"
name="iLaunchAppRequestsProcessed_returned" type="xs:int"/>
                    <xs:element minOccurs="1"
name="iLaunchEmailRequestsProcessed_returned" type="xs:int"/>
                    <xs:element minOccurs="1"
name="iLaunchSmsRequestsProcessed_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="iSIPRequestsProcessed_returned"
type="xs:int"/>
                    <xs:element minOccurs="1" name="iSendAppRequestsProcessed_returned"
type="xs:int"/>
                    <xs:element minOccurs="1"
name="iSendEmailRequestsProcessed_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="iSendSmsRequestsProcessed_returned"
type="xs:int"/>
                    <xs:element minOccurs="1" name="iTELRequestsProcessed_returned"
type="xs:int"/>
                    <xs:element minOccurs="1" name="iVXMLRequestsProcessed_returned"
type="xs:int"/>
                    <xs:element minOccurs="1" name="lastRequestDateTime_returned"
nillable="false" type="xs:dateTime"/>
                    <xs:element minOccurs="1" name="maxMPPHops_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="minMPPHops_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="serviceStartedDateTime_returned"
nillable="false" type="xs:dateTime"/>
                    <xs:element minOccurs="1" name="vpmsSoftwareVersion_returned"
nillable="false" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="SendAppRequest">
                <xs:sequence>
                    <xs:element minOccurs="1" name="applicationName" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="from" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="genericAppType" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="parameters" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="requestTimeout" type="xs:int"/>
                    <xs:element minOccurs="0" name="sessionID" nillable="true"
type="xs:string"/>
                    <xs:element minOccurs="1" name="to" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="0" name="ucid" nillable="true"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="SendAppResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="ID_returned" nillable="false"
type="xs:string"/>
                    <xs:element minOccurs="1" name="parameters" nillable="false"
type="xs:string"/>
                </xs:sequence>
```

```
            </xs:complexType>
            <xs:complexType name="GetStatusRequest">
                <xs:sequence>
                    <xs:element minOccurs="0" name="getStatusRequestUnused"
type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="GetStatusResponse">
                <xs:sequence>
                    <xs:element minOccurs="1" name="avgMPPHops_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="iCCXMLEventsSent_returned"
type="xs:int"/>
                    <xs:element minOccurs="1" name="iCCXMLRequestsProcessed_returned"
type="xs:int"/>
                    <xs:element minOccurs="1" name="iSIPRequestsProcessed_returned"
type="xs:int"/>
                    <xs:element minOccurs="1" name="iTELRequestsProcessed_returned"
type="xs:int"/>
                    <xs:element minOccurs="1" name="iVXMLRequestsProcessed_returned"
type="xs:int"/>
                    <xs:element minOccurs="1" name="lastRequestDateTime_returned"
nillable="false" type="xs:dateTime"/>
                    <xs:element minOccurs="1" name="maxMPPHops_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="minMPPHops_returned" type="xs:int"/>
                    <xs:element minOccurs="1" name="serviceStartedDateTime_returned"
nillable="false" type="xs:dateTime"/>
                    <xs:element minOccurs="1" name="vpmsSoftwareVersion_returned"
nillable="false" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
        <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://io.java/xsd">
            <xs:complexType name="IOException">
                <xs:sequence/>
            </xs:complexType>
        </xs:schema>
        <xs:schema xmlns:ax26="http://services.vp.avaya.com/xsd" xmlns:ax24="http://
rmi.java/xsd" attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://xml.avaya.com/ws/VPAppIntf/VoicePortal">
            <xs:import namespace="http://rmi.java/xsd"/>
            <xs:import namespace="http://services.vp.avaya.com/xsd"/>
            <xs:element name="VPAppIntfServiceRemoteException">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="RemoteException"
nillable="true" type="ax24:RemoteException"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="sendCCXMLEvent">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:SendCCXMLEventRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="sendCCXMLEventResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:SendCCXMLEventResponse"/>
                    </xs:sequence>
                </xs:complexType>
```

```
            </xs:element>
            <xs:element name="queryResources">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:QueryResourcesRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="queryResourcesResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:QueryResourcesResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchVXML">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:LaunchVXMLRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchVXMLResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:LaunchVXMLResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchCCXML">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:LaunchCCXMLRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchCCXMLResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:LaunchCCXMLResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchSMS">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:LaunchSMSRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchSMSResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:LaunchSMSResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
```

```
            <xs:element name="sendSMS">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:SendSMSRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="sendSMSResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:SendSMSResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchEmail">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:LaunchEmailRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchEmailResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:LaunchEmailResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="sendEmail">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:SendEmailRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="sendEmailResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:SendEmailResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
             <xs:element name="launchHtml">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:LaunchHtmlRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchHtmlResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:LaunchHtmlResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchApp">
```

```
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:LaunchAppRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="launchAppResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:LaunchAppResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getStatusEx">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:GetStatusExRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getStatusExResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:GetStatusExResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="sendApp">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:SendAppRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="sendAppResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:SendAppResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getStatus">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="args0" nillable="true"
type="ax25:GetStatusRequest"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getStatusResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="ax25:GetStatusResponse"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getConversation">
                <xs:complexType>
```

```
                    <xs:sequence>
                        <xs:element minOccurs="1" name="args0" nillable="false"
type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getConversationResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="1" name="return" nillable="false"
type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="createConversation">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="1" name="args0" nillable="false"
type="xs:string"/>
                        <xs:element minOccurs="1" name="args1" nillable="false"
type="xs:string"/>
                        <xs:element minOccurs="1" name="args2" type="xs:long"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="deleteConversation">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="1" name="args0" nillable="false"
type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="updateConversation">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="1" name="args0" nillable="false"
type="xs:string"/>
                        <xs:element minOccurs="1" name="args1" nillable="false"
type="xs:string"/>
                        <xs:element minOccurs="1" name="args2" type="xs:long"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getConversationByAlias">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="1" name="args0" nillable="false"
type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getConversationByAliasResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="1" name="return" nillable="false"
type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="addConversationAlias">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="1" name="args0" nillable="false"
type="xs:string"/>
```

```
                        <xs:element minOccurs="1" name="args1" nillable="false"
type="xs:string" maxOccurs="3"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
    </wsdl:types>
    <wsdl:message name="queryResourcesRequest">
        <wsdl:part name="parameters" element="ns:queryResources"/>
    </wsdl:message>
    <wsdl:message name="queryResourcesResponse">
        <wsdl:part name="parameters" element="ns:queryResourcesResponse"/>
    </wsdl:message>
    <wsdl:message name="VPAppIntfServiceRemoteException">
        <wsdl:part name="parameters" element="ns:VPAppIntfServiceRemoteException"/>
    </wsdl:message>
    <wsdl:message name="launchCCXMLRequest">
        <wsdl:part name="parameters" element="ns:launchCCXML"/>
    </wsdl:message>
    <wsdl:message name="launchCCXMLResponse">
        <wsdl:part name="parameters" element="ns:launchCCXMLResponse"/>
    </wsdl:message>
    <wsdl:message name="launchAppRequest">
        <wsdl:part name="parameters" element="ns:launchApp"/>
    </wsdl:message>
    <wsdl:message name="launchAppResponse">
        <wsdl:part name="parameters" element="ns:launchAppResponse"/>
    </wsdl:message>
    <wsdl:message name="launchEmailRequest">
        <wsdl:part name="parameters" element="ns:launchEmail"/>
    </wsdl:message>
    <wsdl:message name="launchEmailResponse">
        <wsdl:part name="parameters" element="ns:launchEmailResponse"/>
    </wsdl:message>
    <wsdl:message name="launchHtmlRequest">
        <wsdl:part name="parameters" element="ns:launchHtml"/>
    </wsdl:message>
    <wsdl:message name="launchHtmlResponse">
        <wsdl:part name="parameters" element="ns:launchHtmlResponse"/>
    </wsdl:message>
    <wsdl:message name="sendAppRequest">
        <wsdl:part name="parameters" element="ns:sendApp"/>
    </wsdl:message>
    <wsdl:message name="sendAppResponse">
        <wsdl:part name="parameters" element="ns:sendAppResponse"/>
    </wsdl:message>
    <wsdl:message name="sendEmailRequest">
        <wsdl:part name="parameters" element="ns:sendEmail"/>
    </wsdl:message>
    <wsdl:message name="sendEmailResponse">
        <wsdl:part name="parameters" element="ns:sendEmailResponse"/>
    </wsdl:message>
    <wsdl:message name="getStatusRequest">
        <wsdl:part name="parameters" element="ns:getStatus"/>
    </wsdl:message>
    <wsdl:message name="getStatusResponse">
        <wsdl:part name="parameters" element="ns:getStatusResponse"/>
    </wsdl:message>
    <wsdl:message name="sendCCXMLEventRequest">
        <wsdl:part name="parameters" element="ns:sendCCXMLEvent"/>
    </wsdl:message>
    <wsdl:message name="sendCCXMLEventResponse">
        <wsdl:part name="parameters" element="ns:sendCCXMLEventResponse"/>
    </wsdl:message>
    <wsdl:message name="getStatusExRequest">
```

```
        <wsdl:part name="parameters" element="ns:getStatusEx"/>
    </wsdl:message>
    <wsdl:message name="getStatusExResponse">
        <wsdl:part name="parameters" element="ns:getStatusExResponse"/>
    </wsdl:message>
    <wsdl:message name="launchSMSRequest">
        <wsdl:part name="parameters" element="ns:launchSMS"/>
    </wsdl:message>
    <wsdl:message name="launchSMSResponse">
        <wsdl:part name="parameters" element="ns:launchSMSResponse"/>
    </wsdl:message>
    <wsdl:message name="launchVXMLRequest">
        <wsdl:part name="parameters" element="ns:launchVXML"/>
    </wsdl:message>
    <wsdl:message name="launchVXMLResponse">
        <wsdl:part name="parameters" element="ns:launchVXMLResponse"/>
    </wsdl:message>
    <wsdl:message name="sendSMSRequest">
        <wsdl:part name="parameters" element="ns:sendSMS"/>
    </wsdl:message>
    <wsdl:message name="sendSMSResponse">
        <wsdl:part name="parameters" element="ns:sendSMSResponse"/>
    </wsdl:message>
    <wsdl:message name="getConversationResponse">
        <wsdl:part name="parameters" element="ns:getConversationResponse"/>
    </wsdl:message>
    <wsdl:message name="createConversationRequest">
        <wsdl:part name="parameters" element="ns:createConversation"/>
    </wsdl:message>
    <wsdl:message name="getConversationRequest">
        <wsdl:part name="parameters" element="ns:getConversation"/>
    </wsdl:message>
    <wsdl:message name="deleteConversationRequest">
        <wsdl:part name="parameters" element="ns:deleteConversation"/>
    </wsdl:message>
    <wsdl:message name="updateConversationRequest">
        <wsdl:part name="parameters" element="ns:updateConversation"/>
    </wsdl:message>
    <wsdl:message name="getConversationByAliasRequest">
        <wsdl:part name="parameters" element="ns:getConversationByAlias"/>
    </wsdl:message>
    <wsdl:message name="getConversationByAliasResponse">
        <wsdl:part name="parameters" element="ns:getConversationByAliasResponse"/>
    </wsdl:message>
    <wsdl:message name="addConversationAliasRequest">
        <wsdl:part name="parameters" element="ns:addConversationAlias"/>
    </wsdl:message>
    <wsdl:portType name="VPAppIntfServicePortType">
        <wsdl:operation name="queryResources">
            <wsdl:input message="ns:queryResourcesRequest"
wsaw:Action="urn:queryResources"/>
            <wsdl:output message="ns:queryResourcesResponse"
wsaw:Action="urn:queryResourcesResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:queryResourcesVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="launchCCXML">
            <wsdl:input message="ns:launchCCXMLRequest" wsaw:Action="urn:launchCCXML"/>
            <wsdl:output message="ns:launchCCXMLResponse"
wsaw:Action="urn:launchCCXMLResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:launchCCXMLVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
```

```
        <wsdl:operation name="launchApp">
            <wsdl:input message="ns:launchAppRequest" wsaw:Action="urn:launchApp"/>
            <wsdl:output message="ns:launchAppResponse"
wsaw:Action="urn:launchAppResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:launchAppVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="launchEmail">
            <wsdl:input message="ns:launchEmailRequest" wsaw:Action="urn:launchEmail"/>
            <wsdl:output message="ns:launchEmailResponse"
wsaw:Action="urn:launchEmailResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:launchEmailVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="launchHtml">
            <wsdl:input message="ns:launchHtmlRequest" wsaw:Action="urn:launchHtml"/>
            <wsdl:output message="ns:launchHtmlResponse"
wsaw:Action="urn:launchHtmlResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:launchHtmlVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="sendApp">
            <wsdl:input message="ns:sendAppRequest" wsaw:Action="urn:sendApp"/>
            <wsdl:output message="ns:sendAppResponse"
wsaw:Action="urn:sendAppResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:sendAppVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="sendEmail">
            <wsdl:input message="ns:sendEmailRequest" wsaw:Action="urn:sendEmail"/>
            <wsdl:output message="ns:sendEmailResponse"
wsaw:Action="urn:sendEmailResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:sendEmailVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="getStatus">
            <wsdl:input message="ns:getStatusRequest" wsaw:Action="urn:getStatus"/>
            <wsdl:output message="ns:getStatusResponse"
wsaw:Action="urn:getStatusResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:getStatusVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="sendCCXMLEvent">
            <wsdl:input message="ns:sendCCXMLEventRequest"
wsaw:Action="urn:sendCCXMLEvent"/>
            <wsdl:output message="ns:sendCCXMLEventResponse"
wsaw:Action="urn:sendCCXMLEventResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:sendCCXMLEventVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="getStatusEx">
            <wsdl:input message="ns:getStatusExRequest" wsaw:Action="urn:getStatusEx"/>
            <wsdl:output message="ns:getStatusExResponse"
wsaw:Action="urn:getStatusExResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:getStatusExVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
```

```
        <wsdl:operation name="launchSMS">
            <wsdl:input message="ns:launchSMSRequest" wsaw:Action="urn:launchSMS"/>
            <wsdl:output message="ns:launchSMSResponse"
wsaw:Action="urn:launchSMSResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:launchSMSVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="launchVXML">
            <wsdl:input message="ns:launchVXMLRequest" wsaw:Action="urn:launchVXML"/>
            <wsdl:output message="ns:launchVXMLResponse"
wsaw:Action="urn:launchVXMLResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:launchVXMLVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="sendSMS">
            <wsdl:input message="ns:sendSMSRequest" wsaw:Action="urn:sendSMS"/>
            <wsdl:output message="ns:sendSMSResponse"
wsaw:Action="urn:sendSMSResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:sendSMSVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="deleteConversation">
            <wsdl:input message="ns:deleteConversationRequest"
wsaw:Action="urn:deleteConversation"/>
            <wsdl:output message="ns:null"
wsaw:Action="urn:deleteConversationResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:deleteConversationVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="getConversation">
            <wsdl:input message="ns:getConversationRequest"
wsaw:Action="urn:getConversation"/>
            <wsdl:output message="ns:getConversationResponse"
wsaw:Action="urn:getConversationResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:getConversationVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="updateConversation">
            <wsdl:input message="ns:updateConversationRequest"
wsaw:Action="urn:updateConversation"/>
            <wsdl:output message="ns:null"
wsaw:Action="urn:updateConversationResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:updateConversationVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="createConversation">
            <wsdl:input message="ns:createConversationRequest"
wsaw:Action="urn:createConversation"/>
            <wsdl:output message="ns:null"
wsaw:Action="urn:createConversationResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:createConversationVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="getConversationByAlias">
            <wsdl:input message="ns:getConversationByAliasRequest"
wsaw:Action="urn:getConversationByAlias"/>
            <wsdl:output message="ns:getConversationByAliasResponse"
wsaw:Action="urn:getConversationByAliasResponse"/>
```

```
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:getConversationByAliasVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
        <wsdl:operation name="addConversationAlias">
            <wsdl:input message="ns:addConversationAliasRequest"
wsaw:Action="urn:addConversationAlias"/>
            <wsdl:output message="ns:null"
wsaw:Action="urn:addConversationAliasResponse"/>
            <wsdl:fault message="ns:VPAppIntfServiceRemoteException"
name="VPAppIntfServiceRemoteException"
wsaw:Action="urn:addConversationAliasVPAppIntfServiceRemoteException"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="VPAppIntfServiceSoap11Binding"
type="ns:VPAppIntfServicePortType">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
        <wsdl:operation name="queryResources">
            <soap:operation soapAction="urn:queryResources" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="launchCCXML">
            <soap:operation soapAction="urn:launchCCXML" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="launchApp">
            <soap:operation soapAction="urn:launchApp" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="launchEmail">
            <soap:operation soapAction="urn:launchEmail" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
```

```
        </wsdl:operation>
        <wsdl:operation name="launchHtml">
            <soap:operation soapAction="urn:launchHtml" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="sendEmail">
            <soap:operation soapAction="urn:sendEmail" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="sendApp">
            <soap:operation soapAction="urn:sendApp" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getStatus">
            <soap:operation soapAction="urn:getStatus" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="sendCCXMLEvent">
            <soap:operation soapAction="urn:sendCCXMLEvent" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getStatusEx">
            <soap:operation soapAction="urn:getStatusEx" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
```

```
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="launchSMS">
            <soap:operation soapAction="urn:launchSMS" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="launchVXML">
            <soap:operation soapAction="urn:launchVXML" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="sendSMS">
            <soap:operation soapAction="urn:sendSMS" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="deleteConversation">
            <soap:operation soapAction="urn:deleteConversation" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getConversation">
            <soap:operation soapAction="urn:getConversation" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
```

```
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="updateConversation">
            <soap:operation soapAction="urn:updateConversation" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="createConversation">
            <soap:operation soapAction="urn:createConversation" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getConversationByAlias">
            <soap:operation soapAction="urn:getConversationByAlias" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="addConversationAlias">
            <soap:operation soapAction="urn:addConversationAlias" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="VPAppIntfServiceSoap12Binding"
type="ns:VPAppIntfServicePortType">
        <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
        <wsdl:operation name="queryResources">
            <soap12:operation soapAction="urn:queryResources" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
```

```
            <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="launchCCXML">
        <soap12:operation soapAction="urn:launchCCXML" style="document"/>
        <wsdl:input>
            <soap12:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="VPAppIntfServiceRemoteException">
            <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="launchApp">
        <soap12:operation soapAction="urn:launchApp" style="document"/>
        <wsdl:input>
            <soap12:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="VPAppIntfServiceRemoteException">
            <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="launchEmail">
        <soap12:operation soapAction="urn:launchEmail" style="document"/>
        <wsdl:input>
            <soap12:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="VPAppIntfServiceRemoteException">
            <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="launchHtml">
        <soap12:operation soapAction="urn:launchHtml" style="document"/>
        <wsdl:input>
            <soap12:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="VPAppIntfServiceRemoteException">
            <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="sendEmail">
        <soap12:operation soapAction="urn:sendEmail" style="document"/>
        <wsdl:input>
            <soap12:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="VPAppIntfServiceRemoteException">
            <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="sendApp">
        <soap12:operation soapAction="urn:sendApp" style="document"/>
```

```
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getStatus">
            <soap12:operation soapAction="urn:getStatus" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="sendCCXMLEvent">
            <soap12:operation soapAction="urn:sendCCXMLEvent" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getStatusEx">
            <soap12:operation soapAction="urn:getStatusEx" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="launchSMS">
            <soap12:operation soapAction="urn:launchSMS" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="launchVXML">
            <soap12:operation soapAction="urn:launchVXML" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
```

```
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="sendSMS">
            <soap12:operation soapAction="urn:sendSMS" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="deleteConversation">
            <soap12:operation soapAction="urn:deleteConversation" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getConversation">
            <soap12:operation soapAction="urn:getConversation" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="updateConversation">
            <soap12:operation soapAction="urn:updateConversation" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="createConversation">
            <soap12:operation soapAction="urn:createConversation" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
```

```
        <wsdl:operation name="getConversationByAlias">
            <soap12:operation soapAction="urn:getConversationByAlias" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="addConversationAlias">
            <soap12:operation soapAction="urn:addConversationAlias" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="VPAppIntfServiceRemoteException">
                <soap12:fault use="literal" name="VPAppIntfServiceRemoteException"/>
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="VPAppIntfServiceHttpBinding" type="ns:VPAppIntfServicePortType">
        <http:binding verb="POST"/>
        <wsdl:operation name="queryResources">
            <http:operation location="queryResources"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="launchCCXML">
            <http:operation location="launchCCXML"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="launchApp">
            <http:operation location="launchApp"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="launchEmail">
            <http:operation location="launchEmail"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="launchHtml">
            <http:operation location="launchHtml"/>
```

```
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="sendEmail">
            <http:operation location="sendEmail"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="sendApp">
            <http:operation location="sendApp"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getStatus">
            <http:operation location="getStatus"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="sendCCXMLEvent">
            <http:operation location="sendCCXMLEvent"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getStatusEx">
            <http:operation location="getStatusEx"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="launchSMS">
            <http:operation location="launchSMS"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="launchVXML">
            <http:operation location="launchVXML"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
```

```
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="sendSMS">
            <http:operation location="sendSMS"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="deleteConversation">
            <http:operation location="deleteConversation"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getConversation">
            <http:operation location="getConversation"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="updateConversation">
            <http:operation location="updateConversation"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="createConversation">
            <http:operation location="createConversation"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getConversationByAlias">
            <http:operation location="getConversationByAlias"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="addConversationAlias">
            <http:operation location="addConversationAlias"/>
            <wsdl:input>
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
```

```
                <mime:content type="application/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="VPAppIntfService">
        <wsdl:port name="VPAppIntfServiceHttpSoap11Endpoint"
binding="ns:VPAppIntfServiceSoap11Binding">
            <soap:address location="http://localhost:8080/axis2/services/
VPAppIntfService"/>
        </wsdl:port>
        <wsdl:port name="VPAppIntfServiceHttpSoap12Endpoint"
binding="ns:VPAppIntfServiceSoap12Binding">
            <soap12:address location="http://localhost:8080/axis2/services/
VPAppIntfService"/>
        </wsdl:port>
        <wsdl:port name="VPAppIntfServiceHttpEndpoint"
binding="ns:VPAppIntfServiceHttpBinding">
            <http:address location="http://localhost:8080/axis2/services/
VPAppIntfService"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Chapter 8: Web Services

Web services

The term **Web services** can be defined as a standardized way of integrating Web-based applications. These **Web services** use technologies such as XML (Extensible Markup Language), SOAP (Simple Object Access Protocol), WSDL (Web services Description Language), and UDDI (Universal Description, Discovery, and Integration). Generally, **Web services** use these technologies as follows:

- XML to tag the data
- SOAP to transfer the data
- WSDL to describe, in standard terms, the services available
- UDDI to list what services are available

For example, a Web service can be created to:

- Look up and return the current value of stocks
- Look up and return the weather forecast for various cities
- Book flight or car rental reservations

In short, **Web services**, like other forms of software, can be created to perform a wide range and variety of functions.

With Orchestration Designer, Web services can be employed within speech, call control, and message flow projects. A Web Service Operation wizard is used to create and define a Web service operation file. This file can be viewed or edited later (for example, to view settings and remap variables) using the Web Service Operation File Editor. You can use the Web service operation file within a speech, call control, or message application.

You can create three types of Web service operation files:

- **Web Service Operation File**: To create Web service operation files based on Axis 1.4. The Axis 1.4 Web service connector is shipped with Orchestration Designer 7.0.1 and above.
- **Web Service Operation File (Axis2)**: To create a Web service operation files based on Axis 2. Axis2 Web service connector is available with Orchestration Designer 7.0.1 and above. Avaya recommends that you use the Axis2 connector for development.

Important:

When selecting a Web service, do not select a service that uses overloading of operations. Overloading of operations is not supported in the SOAP 1.2 standard and is known to cause problems.

Note:

Axis ships with the *Tcpmon* tool for monitoring HTTP traffic. Avaya recommends that you use the Tcpmon tool to debug Web services. Documentation on how to use Tcpmon can be found at: http://ws.apache.org/axis/java/user-guide.html#AppendixUsingTheAxisTCPMonitorTcpmon

- **Web Service Operation File (REST)**: To create Web service operation files based on REST. The **Web Services (REST)** pluggable data connector is available with Orchestration Designer 7.0 and above.

Note:

You can create and use a REST Web service operation file only in a speech project and a message flow project.

Web service headers

When the Web service request is sent to the server, two types of headers can be included with the request; headers included in the HTTP header section or SOAP headers. Both are configured in this section. Each header must be given a unique name and the value for the header is taken from an Orchestration Designer variable.

The direction of the header can be set to IN, OUT or both IN and OUT. If the direction is set to IN, then the value of the header is taken from the Web service response and stored in the selected Orchestration Designer variable. If the direction is set to OUT, then the value of the Orchestration Designer variable is added to a header sent with the Web service request.

Chapter 9: Frequently asked questions about using the Orchestration Designer Report control

For each application, the information you specify in the Orchestration Designer Report control determines what information is displayed in the Avaya Experience Portal Application Detail report and Application Summary report. This topic answers the following questions:

What is an "activity"?

An activity should be comprised of at least two messages, with an **Activity Type** of **Start** and **End** respectively.

The **Start** activity type starts the "duration clock" so that the **Activity Duration** field is populated correctly for each message in the activity. If you do not use a **Start** activity, the duration is always 0.

The **End** activity type stops the "duration clock" so that the **Activity Duration** field is populated correctly for the length of the entire activity.

For example, if you create a Holiday Planning application, it could be broken into 2 main activities: Car Rental and Hotel Booking.

If you have defined a **Start** and **End** time for each activity, you can determine how much time was required to set up the car rental versus book the hotel.

Otherwise, the only information Experience Portal can display is the total length of the call. It cannot help you determine how long it took to process any individual part of that call.

How should "Activity Levels" be used?

The level should be set to something other than **Info** if the message is a result of unusual or undesired flow. The choices are:

- **Fatal**
- **Error**
- **Warning**

For example, when a requested car or hotel is not available, that message should be marked as a **Warning**. If the credit card process did not complete correctly, that message should be marked as either **Error** or **Fatal**, depending on how you want to treat that condition.

This allows you to generate a report showing just those calls in which **Warning**, **Error**, and **Fatal** messages were reported.

Where do I put message details?

The details of a message should be displayed with an associated variable.

For example, if you put the type of car requested in as a variable called `carType` and have a generic Warning-level message stating that a car of that type is not available, Application Detail report users could click on the message to display the value of the associated `carType` variable to find out what specific kind of car was not available.

Furthermore, if you set the activity **Level** filter to **Warning**, the **Variable Name** filter to `carType`, and the **Variable Value** filter to `4Runner`, you can generate an Application Detail report showing all occurrences where a 4Runner was requested but was not available.

How do I specify the value that appears in the Session Label/UCID field in the Application Detail report and Application Summary report?

Assign the value using the **Value** field in the Avaya Properties view.

Note:

This is the only field of the session variable to which you can assign a value. It can contain any arbitrary data which applies to all report messages in the session, such as the account number of a returning customer.

Chapter 10: Resources

Documentation

The following table lists the documents related to Avaya Experience Portal. Download the documents from the Avaya Support website at http://support.avaya.com.

Title	Description	Audience
Avaya Experience Portal Documentation Roadmap	Lists all the documents related to Experience Portal and describes the organization of content across the documents.	Avaya Professional Services Implementation engineers
Administering Avaya Experience Portal	Provides general information about and procedures for administering and configuring specific Experience Portal functions and features using a web-based interface.	Administrators Implementation engineers
Avaya Experience Portal Overview and Specification	Describes tested product characteristics and capabilities, including product overview and feature descriptions, interoperability, performance specifications, security, and licensing requirements.	Administrators Sales engineers Implementation engineers Avaya Professional Services
Implementing Avaya Experience Portal on a single server	Provides procedures to install and configure the Avaya Experience Portal software on a single server.	Implementation engineers
Implementing Avaya Experience Portal on multiple servers	Provides procedures to install and configure Avaya Experience Portal software on two or more dedicated servers.	Implementation engineers

Table continues…

Title	Description	Audience
Deploying Avaya Experience Portal in an Avaya Customer Experience Virtualized Environment	Provides procedures for deploying the Experience Portal virtual application in the Avaya Customer Experience Virtualized Environment. This document includes installation, configuration, initial administration, troubleshooting, and basic maintenance checklists and procedures.	Implementation engineers
Upgrading to Avaya Experience Portal 8.1	Describes how to upgrade your Avaya Experience Portal system to Avaya Experience Portal 8.1.	Implementation engineers
Troubleshooting Avaya Experience Portal	Provides general information about troubleshooting and resolving system problems. This document also provides detailed information and procedures for finding and resolving specific problems.	Administrators Implementation engineers Avaya Professional Services
Avaya Experience Portal Solutions Guide	Provides a high-level description of Avaya Experience Portal as well as topology diagrams, connectivity details, interoperability concept, product interactions, and failover best practices.	Sales engineers Implementation engineers Avaya Professional Services
Avaya Experience Portal Programmer's Reference	Provides information about designing speech applications for Avaya Experience Portal.	Application Developers
Deploying Avaya Experience Portal on Amazon Web Services	Provides procedures for deploying Avaya Experience Portal as *Software as a Solution* by using the Amazon Web Services Management console.	Administrators Implementation engineers Support Personnel Avaya Professional Services
Deploying Avaya Experience Portal on Google Cloud Platform	Provides procedures for deploying Avaya Experience Portal as *Software as a Solution* by using the Google Cloud Platform.	Administrators Avaya Professional Services Implementation engineers Support Personnel

Table continues…

Title	Description	Audience
Deploying Avaya Experience Portal on Microsoft Azure	Provides procedures for deploying Avaya Experience Portal as *Software as a Solution* by using the Microsoft Azure portal.	Administrators Implementation engineers Support Personnel Avaya Professional Services
Avaya Experience Portal Security White Paper	Provides information about the security strategy for Experience Portal, and provides suggestions that companies can use to improve the security of the Experience Portal systems and applications.	Avaya Professional Services Implementation engineers
Avaya Experience Portal Mobile Web Best Practices White Paper	Provides recommended strategies for deploying Avaya Orchestration Designer Mobile Web applications with Avaya Experience Portal, detailing configuration for security, scalability and high availability.	Avaya Professional Services Implementation engineers
Avaya Experience Portal Call Classifications White Paper	Provides information about the call classification feature in Avaya Experience Portal, detailing the configuration and tuning of the call progress engine.	Sales engineers Implementation engineers
Avaya Experience Portal Dialogflow White Paper	Provides information about connecting Avaya Experience Portal self-service applications to Google Dialogflow. This document provides details on configuration, licensing, and other information to help customers with Dialogflow integration.	Avaya Professional Services Implementation engineers
Avaya Experience Portal Nuance Mix Integration White Paper	Provides information about connecting Avaya Experience Portal to Nuance Mix. This document provides details on configuration, licensing, and other information to help customers with Nuance Mix integration.	Avaya Professional Services Implementation engineers

Finding documents on the Avaya Support website

Procedure

1. Go to https://support.avaya.com.

2. To log in, click **Sign In** at the top of the screen and then enter your login credentials when prompted.
3. Click **Product Support** > **Documents**.
4. In **Search Product**, start typing the product name and then select the appropriate product from the list displayed.
5. In **Select Release**, select the appropriate release number.

   This field is not available if there is only one release for the product.
6. **(Optional)** In **Enter Keyword**, type keywords for your search.
7. From the **Select Content Type** list, select one or more content types.

   For example, if you only want to see user guides, click **User Guides** in the **Select Content Type** list.
8. Click to display the search results.

Avaya Documentation Center navigation

For some programs, the latest customer documentation is now available on the Avaya Documentation Center website at https://documentation.avaya.com.

Important:

For documents that are not available on Avaya Documentation Center, click **More Sites** > **Support** on the top menu to open https://support.avaya.com.

Using the Avaya Documentation Center, you can:

- Search for keywords.

  To filter by product, click **Filters** and select a product.
- Search for documents.

  From **Products & Solutions**, select a solution category and product, and then select the appropriate document from the list.
- Sort documents on the search results page.
- Click **Languages** ( ) to change the display language and view localized documents.
- Publish a PDF of the current section in a document, the section and its subsections, or the entire document.
- Add content to your collection using **My Docs** ( ).

  Navigate to the **Manage Content** > **My Docs** menu, and do any of the following:
  - Create, rename, and delete a collection.
  - Add topics from various documents to a collection.
  - Save a PDF of the selected content in a collection and download it to your computer.

  - Share content in a collection with others through email.
  - Receive collection that others have shared with you.
- Add yourself as a watcher using the **Watch** icon ().

  Navigate to the **Manage Content** > **Watchlist** menu, and do the following:

  - Enable **Include in email notification** to receive email alerts.
  - Unwatch selected content, all content in a document, or all content on the Watch list page.

  As a watcher, you are notified when content is updated or deleted from a document, or the document is removed from the website.
- Share a section on social media platforms, such as Facebook, LinkedIn, and Twitter.
- Send feedback on a section and rate the content.

Note:

Some functionality is only available when you log in to the website. The available functionality depends on your role.

Viewing Avaya Mentor videos

Avaya Mentor videos provide technical content on how to install, configure, and troubleshoot Avaya products.

About this task

Videos are available on the Avaya Support website, listed under the video document type, and on the Avaya-run channel on YouTube.

- To find videos on the Avaya Support website, go to https://support.avaya.com/ and do one of the following:
  - In **Search**, type `Avaya Mentor Videos`, click **Clear All** and select **Video** in the **Select Content Type**.
  - In **Search**, type the product name. On the Search Results page, click **Clear All** and select **Video** in the **Select Content Type**.

    The **Video** content type is displayed only when videos are available for that product.

  In the right pane, the page displays a list of available videos.
- To find the Avaya Mentor videos on YouTube, go to www.youtube.com/AvayaMentor and do one of the following:
  - Enter a keyword or keywords in the **Search Channel** to search for a specific product or topic.
  - Scroll down Playlists, and click a topic name to see the list of videos available. For example, Contact Centers.

Note:

Videos are not available for all products.

Support

Go to the Avaya Support website at https://support.avaya.com for the most up-to-date documentation, product notices, and knowledge articles. You can also search for release notes, downloads, and resolutions to issues. Use the online service request system to create a service request. Chat with live agents to get answers to questions, or request an agent to connect you to a support team if an issue requires additional expertise.

Index

A

C

D

E

F

G

H

I

L

M

O

P

Q

R

S

U

V

W