Avaya Dialog Designer

Developer’s Guide

Release 4.1

Notice

While reasonable efforts were made to ensure that the information in this document was complete and accurate at the time of printing, Avaya Inc. can assume no liability for any errors. Changes and corrections to the information in this document may be incorporated in future releases.

Documentation disclaimer

Avaya Inc. is not responsible for any modifications, additions, or deletions to the original published version of this documentation unless such modifications, additions, or deletions were performed by Avaya. Customer and/or End User agree 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 the Customer or End User.

Link Disclaimer

Avaya Inc. is not responsible for the contents or reliability of any linked Web sites referenced elsewhere within this documentation, and Avaya does not necessarily endorse the products, services, or information described or offered within them. We cannot guarantee that these links will work all of the time and we have no control over the availability of the linked pages.

Warranty

Avaya Inc. provides a limited warranty on this product. Refer to your sales agreement 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 through the following Web site:

http://www.avaya.com/support

License

USE OR INSTALLATION OF THE PRODUCT INDICATES THE END USER'S ACCEPTANCE OF THE TERMS SET FORTH HEREIN AND THE GENERAL LICENSE TERMS AVAILABLE ON THE AVAYA WEB SITE AT http://support.avaya.com/LicenseInfo/ ("GENERAL LICENSE TERMS"). IF YOU DO NOT WISH TO BE BOUND BY THESE TERMS, YOU MUST RETURN THE PRODUCT(S) TO THE POINT OF PURCHASE WITHIN TEN (10) DAYS OF DELIVERY FOR A REFUND OR CREDIT.

Avaya grants End User a license within the scope of the license types described below. The applicable number of licenses and units of capacity for which the license is granted will be one (1), unless a different number of licenses or units of capacity is specified in the Documentation or other materials available to End User. "Designated Processor" means a single stand-alone computing device. "Server" means a Designated Processor that hosts a software application to be accessed by multiple users. "Software" means the computer programs in object code, originally licensed by Avaya and ultimately utilized by End User, whether as stand-alone Products or pre-installed on Hardware. "Hardware" means the standard hardware Products, originally sold by Avaya and ultimately utilized by End User.

License Type(s)

Copyright

Except where expressly stated otherwise, the Product is protected by copyright and other laws respecting proprietary rights. Unauthorized reproduction, transfer, and or use can be a criminal, as well as a civil, offense under the applicable law.

Third-party Components

Certain software programs or portions thereof included in the Product may contain software distributed under third party agreements ("Third Party Components"), which may contain terms that expand or limit rights to use certain portions of the Product ("Third Party Terms"). Information identifying Third Party Components and the Third Party Terms that apply to them is available on Avaya's web site at:

http://support.avaya.com/ThirdPartyLicense/

Avaya Fraud Intervention

If you suspect that you are being victimized by toll fraud and you need technical assistance or support, call Technical Service Center Toll Fraud Intervention Hotline at +1-800-643-2353 for the United States and Canada. For additional support telephone numbers, see the Avaya Web site:

http://www.avaya.com/support

Trademarks

Avaya is a trademark of Avaya Inc.

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

Avaya Support

Avaya provides a telephone number for you to use to report problems or to ask questions about your product. The support telephone number is 1-800-242-2121 in the United States. For additional support telephone numbers, see the Avaya Web site:

http://www.avaya.com/support

Contents

About Avaya Dialog Designer Documentation

To help get the most out of Avaya Dialog Designer documentation, take a few moments to review the following topics:

Audience

The Dialog Designer documentation is for those who need to:

- Install and configure Dialog Designer for their own use or the use of others
- Use Dialog Designer to design and create speech applications for:
  - Avaya Interactive Response (IR), Release 1.3 (or later)
  - Avaya Voice Portal 3 (or later)

These users include, among others:

- Customers who want to create their own speech applications
- Avaya business partners and independent service vendors who create speech applications for Avaya customers

The primary users of Dialog Designer are likely to be highly knowledgeable and skilled in telecommunications and Internet technologies. Therefore, this documentation does not cover many topics related to those areas. Instead, Avaya assumes that those using Dialog Designer are already proficient and knowledgeable in the following areas:

- The operating systems in which they will be developing and deploying Dialog Designer applications
- Computer networking concepts and technologies
- Telecommunications concepts and technologies, including switches and gateways
- Basic programming logic and practice

Note:
Although not required to develop applications in Dialog Designer, knowledge of and experience with Java programming is helpful.

Dialog Designer is built on several existing technologies and tools. Because of this, Avaya recommends that Dialog Designer users familiarize themselves with the following technologies, if they are not already familiar with the technologies:

- Eclipse open-source software
- Java servlet technology
- Servlet engine technologies
- Speech recognition and synthesis technologies
- Database administration
- Web service technologies

For more information about and additional resources for these technologies, see Resources.

Resources

Avaya Dialog Designer depends on the use of several closely related software products and technologies. When using Dialog Designer, it is recommended to also review the documentation for these related products and technologies.

Avaya does not reproduce or package the documentation for these related products and technologies. However, to help locate the appropriate documentation, review the following resources:

Note:
All effort was made to ensure that the URLs shown were valid at the time this documentation was published. However, Avaya assumes no responsibility for changed URLs. For more updated URLs, perform a search operation online.

- For Eclipse and supporting Eclipse components (GEF and WTP):

  http://www.eclipse.org/documentation/

  To view the Eclipse online Help in Dialog Designer, click **Help** > **Help Contents**.

- For the Java SDK (Software Developer's Kit):

  http://java.sun.com/j2se/1.5.0/docs/index.html

- For Tomcat: 5.0 or 5.5

  http://jakarta.apache.org/tomcat/tomcat-5.0-doc/

  http://tomcat.apache.org/tomcat-5.5-doc/index.html/

- For IBM WebSphere or WebSphere Express:

  http://www-1.ibm.com/support/docview.wss?rs=180&context=SSEQTP&uid=swg27005148

- WebLogic

  http://commerce.bea.com/showproduct.jsp?family=WLI&major=9.2&minor=1&WT.ac=DL_www_WL-Fam_WLS

- For the Microsoft Speech SDK:

  http://www.microsoft.com/speech/download/sdk51/

- For databases and JDBC implementation:

  http://www.sql.org/

  http://www.firstsql.com/tutor.htm

  http://java.sun.com/developer/onlineTraining/Database/JDBCShortCourse/jdbc/sql.html

- For Web services:

  http://www.w3.org/TR/wsdl

  http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

- For the W3C VoiceXML 2.0 Recommendation:

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

- For the W3C VoiceXML 2.1 Recommendation:

  http://www.w3.org/TR/voicexml21/

- For the W3C CCXML 1.0 Recommendation:

  http://www.w3.org/TR/ccxml/

- For the Speech Recognition Grammar Specification, Version 1.0:

  http://www.w3.org/TR/speech-grammar/#AppJ.5

Documentation Availability

The Avaya Dialog Designer documentation consists of four outputs:

- *Getting Started with Avaya Dialog Designer* – This PDF file contains information needed to install and configure Dialog Designer for initial use, as well as understand the basics of the Dialog Designer GUI.
- *Avaya Dialog Designer online Help* – The online Help provides detailed information and procedures for using Dialog Designer features and options in speech applications

  When installing Dialog Designer, the online Help is installed as an additional Eclipse plug-in.

  Accessing online help can be done in a couple ways, as follows:

  – With the cursor over an item within the interface, press the **F1** key to display help.

  – Click **Help** (menu) > **Help Contents** > **Dialog Designer documentation** > **Dialog Designer Developer's Guide**.
- *Avaya Dialog Designer Developer's Guide* – This PDF document contains the same content available in the online Help system, but in a format that can be printed or viewed using Adobe Acrobat Reader.
- *Programmer Reference Guide* – This online documentation is designed for programmers of Dialog Designer. This guide includes:

  – A **Constants** quick reference guide

  – A **Class Hierarchy** reference guide

  – An **API Reference** guide

  This information is available by clicking **Help** (menu) > **Help Contents** > **Dialog Designer documentation** > **Programmer Reference**.

Where and How to View Documentation

Copies of this documentation are available from several locations, as follows:

- Dialog Designer CD-ROM (which includes the software).
- Downloadable from the Avaya online support Web site at:

  http://support.avaya.com
- Viewable from within Dialog Designer. Click **Help** (menu) > **Help Contents** > **Dialog Designer documentation**.

Chapter 1: About Avaya Dialog Designer

The Avaya Dialog Designer is a Java-based tool for creating speech and/or call control applications that comply with VoiceXML v2.1 or CCXML v1.0 (January 19, 2007 specification). Designed as an Eclipse plug-in, Dialog Designer provides an integrated GUI for the design and implementation of speech applications that can operate with Avaya IR and Avaya Voice Portal systems.

Chapter 2: Installation and Configuration

This chapter describes how to get Avaya Dialog Designer up and running as quickly as possible. In addition, this section includes details on installing sample applications and procedures for upgrading Dialog Designer, or installing patches and updates.

See the following sections:

Ensuring System Requirements Are Met

Before installing Dialog Designer, ensure that the hosting system meets the requirements described in the following sections:

Hardware Requirements

The system that will host the Dialog Designer development environment must meet or exceed the following hardware requirements.

Required Hardware	Minimum	Recommended
CPU speed	1 GHz	2 GHz
RAM	512 MB	1 GB
Hard disk drive	40 GB	n/a
Monitor resolution	1024 x 768 pixels	n/a

Software Requirements

The system that hosts the Dialog Designer development environment must have the following software packages installed, before continuing the Dialog Designer installation and configuration.

Important:
If preparing to upgrade Dialog Designer, see Upgrading Dialog Designer.

Before proceeding with the installation process, ensure that the Dialog Designer development environment host system has the following software installed. These files are all available on the Dialog Designer 4.1 distribution CD.

Supporting Software Requirements

Software Requirement	On CD	Notes and Links[1]
Windows XP (Professional or Home editions, Service Pack 2 or later) Windows Vista (Business or Ultimate editions)[2]	No	Dialog Designer can be installed on any of the defined operating systems as long as the proper hardware requirements are met, and supporting software packages are installed.
J2SE Development Kit 5.0 (JDK 1.5)	Yes	The JDK includes the Java Runtime Environment (JRE) and command-line tools, compilers, and debuggers used in developing applets and applications Install JDK 1.5, as follows: 1. Locate the JDK installer on the Dialog Designer 4.1 CD. This file is located in the following directory: <CD_drive>:\Installation\Java\ 2. Double-click on the downloaded executable file to install the JDK.
Eclipse-Prereq-DD.zip which includes: • Eclipse 3.2 SDK • GEF 3.2 SDK of Eclipse-3.3-Prereq-DD.zip which includes: • Eclipse 3.3 SDK • GEF 3.3 SDK	Yes	Eclipse is a Java-based open-source integrated development environment (IDE) for software development. Dialog Designer runs as an Eclipse plug-in. Dialog Designer also uses this Eclipse GEF (Graphical Editing Framework plug-ins for Eclipse) for advanced graphical functions. Install the Dialog Designer Eclipse Prerequisite files, as follows: 1. Locate the package file on the Dialog Designer 4.1 CD. This file is located in the following directory: <CD_drive>:\Installation\Eclipse\ 2. Extract the ZIP file into an installation folder. 3. If you want, create shortcut for the Eclipse executable. Dialog Designer is launched via Eclipse.

Supporting Software Requirements (continued)

Software Requirement	On CD	Notes and Links[1]
Tomcat 5.0.28 or Tomcat 5.5	Yes	Tomcat generates and serves VoiceXML pages to the Avaya Application Simulator. **Note**: Administrative privileges are required when running Tomcat. Install Tomcat as follows: 1. Locate the Tomcat distribution package on the Dialog Designer 4.1 CD. These files are located in the following directory: <CD_drive>:\Installation\Tomcat\ 2. Extract the ZIP file in a temporary folder. 3. Review the **RUNNING.txt** file for additional installation instructions. **IMPORTANT!** Do not install Tomcat as an NT service. This configuration is not supported because Tomcat does not start and stop appropriately when developing applications.
Microsoft Speech SDK 5.1 (SpeechSDK51.exe)	Yes	Microsoft Speech SDK is used by Dialog Designer during application testing, to perform automated speech recognition (ASR) and Text-to-Speech (TTS) functions. If Microsoft Office 2003 is already installed, check to see if a Speech Recognition and Text-to-Speech tabs are available in the Speech Control Panel. If so, this SDK does not need to be installed. Otherwise, install the Microsoft Speech SDK 5.1, as follows: 1. Locate the Microsoft Speech SDK file on the Dialog Designer 4.1 CD. This file is located in the following directory: <CD_drive>:\Installation\MSSpeech\ 2. Double-click on the **Setup.exe** file, which launches the Microsoft SDK InstallShield Wizard. Click **Next** on the Welcome dialog. 3. Accept the license terms, and click **Next**. 4. Enter a User Name and Organization in the Customer Information dialog, then click **Next**. 5. Accept the default installation folder, when prompted, or navigate to another, if applicable. Then click **Next**. 6. Click **Install** to begin the Microsoft Speech SDK installation. 7. Click **Finish** when the installation is complete.

Supporting Software Requirements (continued)

Software Requirement	On CD	Notes and Links[1]
Eclipse-Prereq-DDCCXML.zip which includes: • WTP 1.5 • emf-sdo-xsd 2.2.0 • JEM 1.2 or Eclipse-3.3-Prereq-DDCCXML.zip which includes: • WTP SDK 2.0 • emf-sdo-xsd 2.3 • DTP 1.5	Yes	This is an optional prerequisite for developers only required if developing Call Control applications in CCXML. If only working with Speech Applications, these files do not need to be installed. Install the Eclipse Prerequisite DDCCXML zip file, as follows: 1. Locate the package file on the Dialog Designer 4.1 CD. This file is located in the following directory: <*CD_drive*>:\Installation\Eclipse\ 2. Extract the file into the directory just above the **eclipse** installation folder. 3. If you want, create shortcut for the Eclipse executable. Dialog Designer is launched via Eclipse. **Note**: This ZIP file is password protected, just as it would be if you downloaded it from Avaya. Go to the Avaya Developer Connection Web site to register for the CCXML feature, and to get a password.
Avaya Dialog Designer WebLM license server	Yes	A valid license is required to run Dialog Designer applications on an Avaya IR, Avaya Voice Portal, or Voice XML platform. Install a license on a new or existing WebLM license server, for the applicable Tomcat version you have already installed (Tomcat 5.0 or 5.5), as follows: 1. Locate the appropriate WebLM installation file on the Dialog Designer 4.1 CD. This file is located in the following directory: <*CD_drive*>:\Installation\WebLM\<*TomcatVersion*>\ 2. To complete the installation, review the WebLM installation/release notes, available on the CD in the following directory: <*CD_drive*>:\Installation\WebLM\

1. Though specific locations on the CD are described here, it is intended that users simply launch the CD, and use the displayed HTML index page to navigate to the required resources.

   By following these instructions, installation will be smoother since the online navigation documentation leads the developer along the correct installation path. This is the preferred methodology for using the CD and installing Dialog Designer efficiently.

2. Requires Eclipse 3.3

Installing Dialog Designer

The Avaya Dialog Designer installation CD contains the Dialog Designer distribution executable. Before running the executable, ensure that all system requirements are met as described in Ensuring System Requirements Are Met.

Note:
Before install Dialog Designer software, it is recommended to temporarily disable antivirus software and close any open or running applications.

Important:
The procedure described in this section is for new installations of the Dialog Designer software. To upgrade Dialog Designer, see Upgrading Dialog Designer.

Install the Dialog Designer software, as follows:

1. Insert the Dialog Designer installation CD in the CD drive, and the Install Wizard starts automatically.

   Note:
   If the Install Wizard does not start up automatically, browse to the root directory of the CD and double-click on **autorun.exe**.

2. An Avaya Software License Agreement is displayed. Review the license agreement, and assuming you accept it, click **I accept. Please continue**.
3. A “start.html” file is displayed, with an overview of the available resources on the CD. The following resources are available:
   - *Getting Started with Dialog Designer* – A link to a copy of this document, on the CD.
   - *Installation Notes* – A link to additional installation notes to get up and running with Dialog Designer right away. These notes include information on installing Dialog Designer and prerequisite software, WebLM information, and localization bundles available in the current release.
   - *Supplements to the Dialog Designer Documentation* – A link to a page describing additional developer resources to assist developers working with Dialog Designer. For example, an online FAQ located at Avaya Developer Connection, Release Notes, and IR 1.3 telephony configuration support details.
   - *Sample Applications* – A link to a page with information about accessing and using sample applications in Dialog Designer. Using sample applications provide tangible ideas on how to use Dialog Designer most effectively.
4. Following the instructions and links in the *Installation Notes* on the CD, install supporting software and also Dialog Designer software as described.

5. Optionally, to install the CCXML feature, you have to install the CCXML prerequisites file: **Eclipse-Prereq-DDCCXML.zip** or **Eclipse-3.3-Prereq-DDCCXML.zip**. For more details, see the Avaya Support website for instructions and download files at: http://support.avaya.com.

This completes the installation procedure. Dialog Designer is now installed, but before it can be used, some basic settings need to be configured. See Configuring Basic Settings to properly configure your development environment settings.

Note:
Avaya recommends that you read the Eclipse “readme” file located in the **/readme** subdirectory where Eclipse is installed. The Eclipse readme file includes valuable information and tips for configuring Eclipse further.

Configuring Basic Settings

Before creating Dialog Designer projects and after installing all required software, perform the basic configurations described in the following sections to ensure that the environment is properly configured and ready to use:

- Creating Eclipse Shortcut
- Setting Up the Workspace
- Setting Dialog Designer Preferences
- Configuring the Microsoft Speech SDK for Microphone Input

Creating Eclipse Shortcut

Dialog Designer is accessed by launching Eclipse after all installation components have been installed. To readily access Eclipse, Avaya suggests creating a Windows desktop shortcut icon to the Eclipse executable file (**eclipse.exe**) located where Eclipse is installed.

To launch Dialog Designer, double-click on the created shortcut file, which opens the Workspace Launcher dialog.

Setting Up the Workspace

Upon first launching Avaya Dialog Designer (via Eclipse), the Eclipse **Workspace Launcher** dialog prompts for a workspace location to be designated. This directory is where Dialog Designer project files will be saved.

Important:
If configuring a new version of Dialog Designer, make backup copies of all files in the original installation directory before configuring a new directory.

The default directory is relative to the installation path of Eclipse (for example, **C:\Eclipse\workspace**). Click **Browse** to navigate to a different directory, if desired.

To stop the **Workspace Launcher** dialog from prompting for this directory with every launch of Eclipse, check the **Use this as the default and do not ask again** option on the dialog.

Setting Dialog Designer Preferences

Dialog Designer preferences must be configured the first time that Dialog Designer is used. Subsequent launches of Dialog Designer will assume these same configured preferences.

To access these preference settings, from the **Window** menu, select **Preferences**. Verify or configure the preference settings as described.

Dialog Designer Preferences Settings

Preferences Area	Notes

Perspectives	In the selection pane on the left, select **General > Perspectives**. Configure the following options should be selected:
	• In the **Open a new perspective** panel, select **In the same window**.
	• In the **Open a new view** panel, select **Within the perspective**.
	• In the **Open the associated perspective when creating a new project - Prompt** panel, select **Prompt**.
	• In the **Available perspectives** panel:
	  • Select either **Speech** or **Call Control** as your default perspective, depending on the type of work you are doing.
	  • Click **Make default**.

Dialog Designer Preferences Settings (continued)

Preferences Area	Notes
Tomcat	In general, if Tomcat was installed with default settings, the following settings will already be configured. If you used the default installation location for a Tomcat 5.0 installation, for example, the path is: **C:\Program Files\Apache Software Foundation\Tomcat 5.0** where *C:* is the drive letter of your primary drive. **Tomcat home** is the path where Tomcat is installed. For example: **C:\Program Files\Apache Software Foundation*Tomcat 5.X*** The default **Contexts directory** for example is: **C:\Program Files\Apache Software Foundation*Tomcat 5.X*\conf\Catalina\localhost** where *C:* is the drive letter of your primary drive. The correct path must take you to the **...\Catalina\localhost** directory. **Note:** If you are just running Dialog Designer only in your development environment (that is, not running deployed applications), you do not have to install the runtime config to your local Tomcat; this is handled automatically. You only need to set up your production system when you are deploying and running live applications.
Dialog Designer	If your design environment is behind a firewall (only), proxy settings are required, as follows: ***Proxy Settings*** • **Enable HTTP proxy connection** – If a proxy server for Internet access is required, select this check box. • **Ignore hosts with addresses** – To allow Dialog Designer to ignore HTTP hosts with specific addresses, specify the IP address(s). For multiple addresses, use either a comma or semi-colon for a separator character. • **HTTP proxy host address** – Enter the full HTTP path to, or URL of, the proxy server. • **HTTP proxy host port** – Enter the port that Dialog Designer can use to access the proxy server. ***Copy HTTP Settings to HTTPS*** • Click this button to copy the configured HTTP settings to HTTPS settings automatically.

Dialog Designer Preferences Settings (continued)

Preferences Area	Notes
Dialog Designer (continued)	***HTTPS Proxy Settings*** • **Enable HTTPS proxy connection** – If a proxy server for Internet access is required, select this check box. If a proxy server is not required for Internet access, clear this check box. If cleared, Dialog Designer disables the other options in this area. • **Ignore HTTPS hosts with addresses** – To allow Dialog Designer to ignore HTTPS hosts with specific addresses, specify the IP address(s). For multiple addresses, use either a comma or semi-colon for a separator character. • **HTTPS proxy host address** – Enter the full HTTPS path to, or URL of, the proxy server. If you do not know this address, look at the proxy server settings for your Internet browser software. • **HTTPS proxy host port** – Enter the port that Dialog Designer can use to access the HTTPS proxy server. ***Runtime License Server*** If the deployment application server requires a license server to use speech resources, such as ASR or TTS, provide the URI to the home page of that license server. **Note**: Specifying a runtime license server is only needed if you have a platform (Voice Portal or IR) accessing your application from the development environment. The format for this URI is: **http://*webServerName*:*port*** where: • ***webServerName*** is the fully qualified domain name of your WebLM license server • ***port*** is the number of the port the system uses to access the license server For example: **http://licenseServer.myCompany.com:8080** If you do not know the URI, contact your Avaya technical service representative. **Note**: These settings are required even if proxy options are also set in Microsoft Internet Explorer or any other Web browser.
Java	In the Installed JREs sub-options, verify that **jre1.5.x** JRE is selected. If **jre1.5.x** JRE does not appear in the list, click **Add...** to add it. **Note**: The Compiler sub-option should have a JDK Compliance value of 5.0

Configuring the Microsoft Speech SDK for Microphone Input

Dialog Designer uses the Microsoft Speech SDK to handle ASR input from a microphone during application simulation. To use the Microsoft Speech SDK for ASR input, the SDK must be configured to use a microphone.

Configure the Microsoft Speech SDK for using a microphone, as follows:

1. On the system where Dialog Designer is installed, access the **Control Panel**.
2. Double-click the **Speech** option. The **Speech Properties** dialog is opened.
3. In the Speech Recognition tab, with a microphone plugged in and turned on, speak into the microphone. The **Level** indicator in the Microphone panel should show that the system is receiving microphone input. If it does not, click **Audio Input** and correct the audio input source settings.
4. Click **Configure Microphone** to further tune the microphone settings.
5. When done, click **OK**.

Installing Sample Applications

Dialog Designer includes numerous sample applications that can be used to see how finished applications work and operate and to examine how the many features of Dialog Designer work.

Install and run sample applications, as follows:

1. Navigate to the **/Sample Applications/files** directory on the Dialog Designer installation CD.
2. Locate and double-click the **DialogDesigner [*version*] Sample Applications.doc** file.
3. Follow the instructions in that document to finish installing and configuring Dialog Designer sample applications. Also, for detailed information about each of the sample applications available for reference.

Important:
Included sample applications are intended to be used as technical samples for reference only, and not production-ready applications.

Upgrading Dialog Designer

The procedure to upgrade Dialog Designer 4.0 to Dialog Designer 4.1 is slightly different depending on whether the prior version (4.0) will be kept available while installing a concurrent new version (4.1), or not.

See the following sections:

- If Maintaining a 4.0 Environment Is Necessary
- If Preserving a 4.0 Environment Is Not Necessary

See the following sections for additional information:

- Ensuring System Requirements Are Met
- Installing Dialog Designer

If Maintaining a 4.0 Environment Is Necessary

If it is necessary to keep a Dialog Designer 4.0 environment (for example, to continue maintenance of 4.0-based applications), Dialog Designer 4.1 can be installed into a separate directory.

In this case, the following considerations should be reviewed:

1. Preserve the Dialog Designer 4.0 and Eclipse 3.2 installation and workspace. Dialog Designer 4.0 will continue to use the previous Tomcat installation.
2. Install Dialog Designer 4.1, and Tomcat 5.5 (Tomcat upgrade is optional) to a separate location (for example, **C:\DD4.1**)

   (Tomcat can also be installed under the Dialog Designer 4.1 installation location to keep it separate.)
3. To upgrade a Dialog Designer 4.0 application to Dialog Designer 4.1:
   - Copy the project from the Dialog Designer 4.0 workspace to the Dialog Designer 4.1 workspace.

     Keeping a copy in the 4.0 workspace ensures that you have a backup in the event that upgrade problems are encountered.
   - Import the copied project into Dialog Designer 4.1. The project will be converted for Dialog Designer 4.1. This project can no longer be opened in Dialog Designer 4.0.
4. If you use a source control system, store the 4.1 application in a different location (or a different branch) so that the old 4.0 application can be maintained in the future.

Following are example (recommended) installation paths for multiple Dialog Designer/Eclipse versions. (base) means any parent directory.

c:\(base)\DD4.0

eclipse	(Eclipse 3.2 install, with GEF 3.2 and Dialog Designer 4.0 features)
tomcat	(Tomcat 5.0 for running Dialog Designer 4.0 applications
workspace	(Dialog Designer 4.0 projects)

c:\(base)\DD4.1

eclipse	(Eclipse 3.2 or 3.3 install, with GEF 3.2 or 3.3 respectively, and Dialog Designer 4.1 features)
tomcat	(Tomcat 5.0 or 5.5 for running Dialog Designer 4.1 applications)
workspace	(Dialog Designer 4.1 projects)

This is a convenient installation structure that keeps the Dialog Designer 4.0 and Dialog Designer 4.1 environments and prerequisite software separate.

Note:
After creating a new workspace during an upgrade, go to Window > Preferences to configure your preferences before importing old projects. See Setting Dialog Designer Preferences for more details.

If Preserving a 4.0 Environment Is Not Necessary

If it is not necessary to preserve a Dialog Designer 4.0 environment, install Dialog Designer 4.1 as follows:

1. Before uninstalling Dialog Designer 4.0, make a backup copy of the projects in your workspace.
2. Uninstall Dialog Designer 4.0, using the **Add or Remove Programs** Control panel option. Depending on the location of your workspace, your projects may be removed by the uninstall process.
3. Install Dialog Designer 4.1 and associated supporting software requirements. See Ensuring System Requirements Are Met.
4. Copy the Dialog Designer 4.0 projects (from the backup) into the Dialog Designer 4.1 workspace.

   It is a good idea to keep your old backup copy in the event that there are errors upgrading.
5. Import each project into Dialog Designer 4.1. The project will then be converted for Dialog Designer 4.1.

6. If you use a source control system, create a branch or store the 4.1 application in a different location so that the old 4.0 application can be maintained, if necessary, in the future.

Note:
After creating a new workspace during an upgrade, go to Window > Preferences to configure your preferences before importing old projects. See Setting Dialog Designer Preferences for more details.

Installing Dialog Designer Patch Updates

At this time, Avaya does not automatically alert you to the availability of new patches to Dialog Designer. Therefore, periodically check the Avaya support Web site for the availability of patches. Or, as an alternative, use the Eclipse update mechanism to check for available updates.

The following sections describes steps for installing a Dialog Designer patches:

- Before Installing a Patch Update
- Installing a Patch Update

Note:
The procedure described in the section is for installing patches or updates to released software, not upgrading software versions completely. For upgrading the software, see Upgrading Dialog Designer.

Before Installing a Patch Update

Before installing a patch update, make backup copies of all files in the default **/eclipse** installation directory, as well as all files in the designated **/workspace** directory (if not a subdirectory of **/eclipse**). This is important in case an update needs to be reverted for any reason.

Also, consider whether to keep and continue to use the older version for existing applications. In this case, Avaya recommends installing the new version using a “clean installation” in a new directory.

CAUTION:
When opening an application created with a prior release of Dialog Designer, Dialog Designer prompts to update the project to the new version. Consult the Release Notes for project conversion considerations.

Installing a Patch Update

Avaya Dialog Designer patch updates are released via Avaya's Support Website. These updates can be accessed directly from within the Eclipse user interface as follows:

1. From the **Help** menu, select **Software Updates** > **Find and Install**.

Note:
If you cannot connect to the update site, check to make sure that proxy settings are properly configured.

2. In the **Install/Update** dialog, select the option **Search for updates of the currently installed features**.
3. Click **Next**.

   Dialog Designer automatically checks both the Eclipse and Avaya Support Web sites for updates. If patches or updates are found, the search mechanism returns the results. Select the updates or features to install and then follow the prompts.

CAUTION:
The Eclipse Install/Update mechanism can be used by other features besides Dialog Designer. To avoid installing incompatible features, it is recommended to only update Dialog Designer features.

If you are not sure which updates to install or have questions about the installation procedure, contact Avaya support at http://support.avaya.com.

Chapter 3: Getting Familiar with the Dialog Designer User Interface

This chapter describes how to access the *Eclipse Workbench User Guide* for more details on Eclipse development environment concepts and terminology used by Avaya Dialog Designer.

In addition, the Dialog Designer user interface (within the Eclipse framework) is described, including different areas of the Dialog Designer workbench, menus, and toolbar options are all described.

See the following sections:

Accessing the Eclipse *Workbench User Guide*, “Concepts” Section

Eclipse Workbench User Guide presents an overview of many of the same concepts used within Dialog Designer, but from the Eclipse development environment framework perspective. Nonetheless, reviewing this information is valuable as a first step in getting familiar with the Eclipse user interface.

Access the *Eclipse Workbench User Guide* online help, as follows:

1. Within Dialog Designer, access the **Help** menu, and then click **Help Contents**. The system displays the *Help - Eclipse Platform* window.
2. In the **Contents** navigation pane, click **Workbench User Guide**.
3. Double-click **Concepts**. Review the multiple sections of the Concepts section to gain familiarity with Eclipse.

Dialog Designer Workbench

Note:
This section assumes familiarity with Eclipse concepts and terminology. See Accessing the Eclipse Workbench User Guide, “Concepts” Section.

The Dialog Designer workbench was designed as a *speech project* perspective in Eclipse. The layout of the views and workspace are optimized to assist you in the creation of speech application projects.

The following descriptions provide an understanding of how the Eclipse environmental elements are arranged in Dialog Designer and why. However, as with any Eclipse perspective, the user is free to arrange perspective elements as desired, but if this occurs, the following view descriptions may no longer be applicable.

The following sections describe views available in the standard Dialog Designer speech perspective. See Dialog Designer Menu and Toolbar Options for details on menus and toolbar items.

Starting in the upper-left corner of the window and working left-to-right and top-to-bottom, the Dialog Designer speech perspective consists of the following major elements:

- Navigator View
- Editor View
- Outline View
- Avaya Application Simulator View
- Problems View
- Tasks View
- Eclipse Properties View
- Console View

Most views or tab groups within the Eclipse perspective can be manipulated by right-clicking on a tab area (on top of the tab label), then selecting any of the available options. The options are:

- **Fast View** - Minimizes the selected tab view, which is accessible via a clickable icon in the lower right corner, for access at a “fast view” access at a later time.
- **Detached** - Allows for the selected tab view to be detached from the Speech perspective, much in the same way that Move does.
- **Restore** - Restores the default settings of the selected tab view. Alternatively, select **Window > Reset Perspective**, which also returns the tab view settings to the default positions and sizes.
- **Move** - Allows the tab view, or complete tab group (if more than one) to be moved outside of the perspective (such as to the Desktop area outside of the Eclipse window).

- **Size** - Adjust the size of the panel border relative to the option selected. Only borders that can be adjusted, per the selected tab, are active options. Options are Left, Right, Top, Bottom. For example, in the Navigator tab, to make the area wider, select Size > Right, and then drag the dark blue highlighted border to make the view wider.
- **Minimize** - Minimize the size of the tab view selected.
- **Maximize** - Maximize the size of the tab view selected.
- **Close** - Close the tab view that is selected.

Navigator View

Located by default in the upper-left corner of the window, the **Navigator** view is a standard Eclipse view. It provides a hierarchical view of the resources in the Dialog Designer speech project.

Of special interest to Dialog Designer users:

- When you create a speech project in Dialog Designer, it automatically creates the following standard folders in the project folder:
  - **connectivity** - This folder contains any database or Web service operations files you create as part of your project.
  - **data** - This folder contains mostly files that are derived from other files. For example, when you create a grammar, the Grammar File Editor creates a ***.gram** file that contains the metadata for the grammar. When you generate the project, Dialog Designer creates a ***.grxml** file that contains the actual XML grammar file. This ***.grxml** file is derived from the ***.gram** file.

    Usually, the derived files in this folder are hidden, and are not shown in the **Navigator** view.

    Important:
    Do not attempt to manually edit these files.

    This folder also contains the **log** folder. The **log** folder is where log files created during application testing are written. You can view these log files, if necessary, as an aid in debugging your applications.

    Finally, this folder also contains a **temp** folder where any messages you record during application testing are stored.
  - **flow** - This folder contains the **main.flow** file. This file is the core of your call flow application (speech project). It is built and edited using the Call Flow Editor.
  - **icons** - This folder contains several icons used for Dialog Designer. Do not delete or alter these icons.

  - ***languageName*** - This folder contains all the phrase, prompt, and grammar definition files for your speech project. Whenever you create or import a phrase file, a prompt file, or a grammar file, Dialog Designer places the file in the appropriate subfolder of this folder.

    The actual name of this folder is the name assigned to the project primary language.
  - **WEB-INF** - This folder contains all the output files created whenever you generate or build the project. Essentially, these files are what get packaged when preparing the application for deployment.
  - **work** - This folder is used as a temporary folder by the Tomcat servlet engine. Do not attempt to manually edit the contents of this folder.

- You can open the editors for many project resources, such as phrases, prompts, and grammars by double-clicking the file in the **Navigator** view. For example, to open a specific phrase in the Phrase File Editor, you can double-click the ***.phrase** file.
- You can also perform many other actions on selected project resources by right-clicking the desired resource and then, selecting the appropriate action from the context menu.

Tip:
One especially useful option of these context menus is the ability to generate individual files or project resources. For example, to regenerate a project grammar without having to regenerate the entire project, right-click on the *.gram file in the **Navigator** view. Then click **Generate** from the context menu.

For more information about the Navigator view in Eclipse see the topic "Navigator view" in the Eclipse *Workbench User Guide* online Help.

Editor View

By default, the Editor view is located in the upper-right area of the window. The main area of this view is called the *workspace*. The workspace is where most of the development work for building a speech application project takes place.

Within the Editor View, there are numerous “sub”-editors that can be launched for creating, updating and managing the development of your speech application.

See the following sections:

- Dialog Designer Editors
- Editor View Tabs

Dialog Designer Editors

Following is a high-level overview of the editors available in Dialog Designer:

- **Call Flow Editor** - The primary editor for Dialog Designer. In the Call Flow Editor, nodes are placed and connected, which create the call flows that direct the caller experience.
- **Phraseset File Editor** - Used to modify phrase set metadata, such as phrase text, relevant comments, and search keywords. Phrase sets can also be recorded and saved in ***.wav** file format.
- **Prompt File Editor** - Used to define and modify prompts, from very simple announcements to very complex prompts involving variables, conditions, and logic.
- **Variable Editor** - Used to create, define, and modify variables, and to view the variables that exist in the current project.
- **Grammar File Editor** - Used to define custom or select built-in grammar files.
- **Database Operation File Editor** - Used to define and modify the way a project works with selected databases.
- **Web Service Operation File Editor** - Used to define and modify the way a project works with selected Web services.
- **Event Type Editor** (available only from within the Call Flow Editor) - Used to define and modify custom events and how they are implemented within a project.
- **CCXML File Editor** - Used to modify CCXML files that are part of Call Control projects.
- **JSP File Editor** - Text-based editor for modifying JSP files that are part of Call Control projects.

Editor View Tabs

Dialog Designer (Eclipse) uses two types of tabs in an Editor view:

- **Editor tabs (or top tabs)** - Whenever an editor is opened, the Editor view displays a tab for that editor in the view toolbar. Each tab displays an icon representing the type of editor associated with the tab, along with the name of the project element currently open.

  Any editor can be closed by clicking the **X** on the tab. When the editor closes, the Editor view no longer displays the tab.

  Note:
  One exception is the Event Type Editor. To close this editor, click the small **x** in the upper-right corner of the Call Flow Editor workspace.

- **Page tabs (or bottom tabs)** - Some editors are multiple page editors. That is, they have more than one workspace page associated with them. In such cases, the Editor view displays a *page tab* at the bottom of the view for each workspace page available in the active editor.

  For example, when the Phrase File Editor is active, the **Phrase** and **Audio** page tabs both become available along the bottom toolbar of the Editor view.

Outline View

Located by default in the lower-left corner of the window, the **Outline** view is a standard Eclipse view. In Dialog Designer, the **Outline** view functions only in the following conditions:

- When the Call Flow Editor is the active editor
- When a Java (***.java**) file is being edited.

There are three different views that can be selected by clicking on the three icon button in the upper right of the Eclipse window, as follows:

- First icon provides a *Node List view*. It lists the nodes in the call flow in alphabetical order and also lists all symbolic node references.
- Second icon provides the *Thumbnail view*. If the call flow is complicated, it allows you to jump around easier within the flow. By clicking and dragging the shaded area in the **Outline** view, the main workspace simultaneously updates to present the full size view of the shaded area.
- Third icon provides the *Bookmark View*, to display a list of all the bookmarks in a call flow. This list can be used to navigate between bookmarks in a call flow which is especially useful with large complex call flows.

  To use the **Bookmark** option in the **Outline** view, click the **Bookmark** icon in the **Outline** view toolbar. Dialog Designer displays a list of all the bookmarks in your call flow. Navigate to a particular bookmark by clicking the bookmark in the main workspace. The focus in the main workspace of the Call Flow Editor shifts to display the clicked bookmark.

Avaya Application Simulator View

Unique to Dialog Designer, the **Avaya Application Simulator** view offers a display and controls for the Avaya Application Simulator, a VXML and CCXML interpreter program.

Dialog Designer uses the Avaya Application Simulator to test applications by simulation. The view includes the following tabs that present different aspects of the browser's operation:

- Application Tab
- Call Tab (call_name Tab)

- CCXML Log Tab
- VXML Log Tab
- Connector Log Tab
- Script Tab
- Parameters Tab

By default, Dialog Designer displays the Avaya Application Simulator view in a tabbed notebook, on top of the **Problems** view and the **Tasks** view, in the lower-center area of the window.

Application Tab

The **Application** tab is the primary (default) tab for the Avaya Application Simulator. Use this tab to start or end the simulation of the project and to set pre-simulation conditions.

This tab includes the following options:

Avaya Application Simulator View—Application Tab Buttons/Features

Button/Feature	Description
Available Projects	Select the project to simulate. If multiple projects are available, select only one at a time.
Calling Number	Simulates ANI. Enter the number that represents the originating telephone number in this field. Use numbers only. **Note:** If you use characters other than numbers, the Avaya Application Simulator passes them along exactly as you enter them in this field. However, in a live system with a deployed application, only numbers are passed to the application. For this reason, Avaya recommends that you use only numbers in this field during simulations.
Called Number	Simulates DNIS. Enter the number that represents the telephone number the caller must dial to get to this application. Use numbers only. **Note:** If you use characters other than numbers, the Avaya Application Simulator passes them along exactly as you enter them in this field. However, in a live system with a deployed application, only numbers are passed to the application. For this reason, Avaya recommends that you use only numbers in this field during simulations.
Converse On Data	Lets you simulate data that you can receive from the switch. **Note**: The application must be specifically designed to collect converse on values. For setting up converse on data on the actual runtime platform requires configuration on both the switch and vectors as well as the IR or Voice Portal. Refer to IR or Voice Portal documentation for working with converse on data.

Avaya Application Simulator View—Application Tab Buttons/Features (continued)

Button/Feature	Description
Input Parameters	Used to simulate input expected from another application module. The format of this simulated input must match the expected format.
Run Application	Click this button to start the simulation. This button is active only when a project is selected for simulation. **Note:** If there are any errors in the project to be simulated, this button is inactive.
End Application	Click this button to end the simulation before it is finished running. This button is active only when a project is in simulation mode. **Note:** This action is different from that of the **Hang Up** simulation button on the **Input** tab (*call_name* tab). This button stops the application wherever it is when you click it. The **Hang Up** button on the **Input** tab simulates a caller being disconnected either by hanging up or some other means.
Do Not Run As Module	Lets you run Dialog Designer applications that are modules as standalone applications. Otherwise, it generates a default VXML page to invoke the Dialog Designer application as a subdialog.

Call Tab (*call_name* Tab)

Note:
Call tabs are added dynamically during application simulation, and therefore not visible until running an application. For CCXML applications, it is possible to have more than one call appearance, and thus more than one Call tab.

The Avaya Application Simulator (AAS) uses the **Call** tab, or rather the *call_name* Tab, to display progress and collect response input during application simulations. When you start a call, the tab is dynamically created (for each call that the simulator is handling).

Use this tab to provide simulated caller responses and set up run-time error conditions. Error conditions include incorrect or no responses from callers, unexpected caller hang-ups, and other potential problem situations.

The Call Tab includes the following options:

Avaya Application Simulator View—Call Tab (*call_name* Tab) Buttons/Features

Button/Feature	Description
Telephone keypad	Use the twelve keys on the left side of the view to simulate caller DTMF (touchtone) input. When pressing (clicking) the keys to respond, the Avaya Application Simulator displays the characters in the field just beneath the keypad.

Avaya Application Simulator View—Call Tab (*call_name* Tab) Buttons/Features

Button/Feature	Description
Send Digits	After pressing response keys in the simulated telephone keypad, click this button for the simulation to continue.
Call Active	Indicates the status of the call simulation, as follows: ● Green – a call simulation is in progress ● Red – a call simulation is not in progress ● Yellow – a Dialog Designer application is running within the CCXML perspective in the simulator
Waiting ASR	When the status is green, the simulation is waiting for a spoken response. (However, when barge-in is enabled, the green light will not appear until you speak.) **Note:** To simulate a spoken response, the Microsoft Speech SDK must be installed and configured properly. The Microsoft Speech SDK should have been installed during the Dialog Designer installation process. To configure the SDK for microphone input, see Configuring the Microsoft Speech SDK for Microphone Input.
Waiting DTMF	When the status is green, the simulation is waiting for a DTMF response. To simulate a DTMF response, click the keys on the keypad that correspond with the response expected from the caller, and then click **Send**.
Input Result	Indicates the type of input result. Choose from the following: ● **No Input** – Simulate no response from a caller. ● **No Match** – Simulate a response from a caller that does not match expected input. ● **Send ASR** – Send an ASR.
Record End	This button is active only when the simulation requires a caller to respond by recording something. Since Avaya Application Simulator does not detect a terminating silence, use this button to simulate a terminating silence and signal the response recording is done. **Tip:** To simulate the use of DTMF key presses to end recordings, set the **DTMF Terminate** property of the **Record** item to **True**. Then, to simulate this, use the telephone keypad to click touchtone keys and click **Send**.
Hang Up	Click this button to simulate a caller hanging up. In this case, the application proceeds to the next node.

Avaya Application Simulator View—Call Tab (*call_name* Tab) Buttons/Features

Button/Feature	Description
Call Classification	These parameters allow you to simulate different types of calls at the voice platform for outcalling call classification in CCXML. The types of calls are: • busy_tone - busy tone • reorder - fast busy tone • sit_tone - failure on the receiver's end • live_voice - person answered • recorded_msg - answering machine answered call with recorded message • msg_end - message terminated, recording ended (only received after recorded_msg) • fax_answer_tone - receiving tone, outbound call to a fax machine • timeout - timeout on classification expired • error The call classifications are passed to an application via the "call classification" query string variable to the Web Service.
Call Status	Simulates the results of a call transfer action. The drop-down list becomes available only when the application contains a **Bridged Transfer** node. Result options that can be simulated include: • **Ring No Answer** - Simulates no answer from the party being called. A value of **noanswer** is assigned to the **Value** field of the applicable Bridged Transfer variable, which can then be used to redirect the call flow. In deployed applications, the **Connect Timeout** property of the **Bridged Transfer** item determines the amount of time the system waits for an answer before it returns control of the call to the application. • **Busy** - Simulates a busy signal for the line of the party being called. A value of **busy** is assigned to the **Value** field of the applicable Bridged Transfer variable, which can be used to redirect the call flow. • **Call Fail Result** • **No Resource** • **Network Busy** • **No Route** • **Unreachable**
Avaya Application Simulator progress display	On the right side of this browser tab, there is a display-only pane which shows progress messages that are generated as the simulation proceeds. These messages tell you what the Avaya Application Simulator is doing as it simulates the execution of the application. This can be particularly helpful when debugging applications.

CCXML Log Tab

The **CCXML Log** tab is a read-only display of the activity log created during simulations. This activity log records the output from the Avaya Application Simulator.Use the logged information to help debug applications, particularly if creating custom Call Control XML code.

VXML Log Tab

The **VXML Log** tab is a read-only display of the activity log created during simulations. This activity log records the output from the Avaya Application Simulator.Use the logged information to help debug applications, particularly if creating custom VoiceXML code.

Connector Log Tab

This tab contains a view of the log output from the IC and CTI connectors.

Script Tab

As an alternative to simulating caller responses and other inputs during simulation, create an XML script to simulate them automatically running a simulation. Scripts can be used to simulate IC and CTI connector functions.

Following are some general command guidelines and examples of scripts to simulate caller response.

```
/*
* Read an xml script and validate the commands and send them to
* the AVB as requested. See the AvayaVXIRef Release notes for details
* on the commands.
*
* Below is a list of all the valid commands. Note that
* the "value" for the rec asr will vary as will the value
* for rec dtmf, record err, and record dtmf.
*
* For a rec type asr you can have one or more items. Using
* multiple items allows you to simulate n-best results.
*
* <?xml version="1.0" encoding ="UTF-8"?>
* <callscript>
* <call>
* <command name="rec" type="asr">
* <item value="dog blue"/>
* </command>
* <command name="rec" type="asr">
* <item value="dog blue" confidence=".7"/>
* </command>
```

```
* <command name="rec" type="asr">
* <item value="dog blue"/>
* <item value="fog blue"/>
* </command>
* <command name="rec" type="asr">
* <item value="dog blue" conficence=".7"/>
* <item value="fog blue" conficence=".4"/>
* </command>
* <command name="rec" type="dtmf" value="1" />
* <command name="rec" type="err" value="nomatch" />
* <command name="rec" type="err" value="noinput" />
* <command name="tel" type="hangup" delay="10"/>
* <command name="tel" type="answered"/>
* <command name="tel" type="answered" value="LIVE VOICE"/>
* <command name="tel" type="progress" value="some reason value"/>
* <command name="tel" type="busy"/>
* <command name="tel" type="noanswer"/>
* <command name="record" type="max"/>
* <command name="record" type="done"/>
* <command name="record" type="err" value="45" />
* <command name="record" type="dtmf" value="1" />
* <command name="quit" type="" value="" />
* <command name="merge" type="" value="ok" />
* <command name="merge" type="" value="fail" />
* </call>
* </callscript>
*
*
* Example of handling a transfer, far end hangs up after 10 seconds.
*
* <?xml version="1.0" encoding ="UTF-8"?>
* <callscript>
* <call>
* </call>
* <call>
* <command name="tel" type="answered" value="LIVE VOICE"/>
* <command name="tel" type="hangup" delay="10"/>
* </call>
* </callscript>
*
*
* Example of handling a transfer, near end hangs up after 10 seconds.
*
* <?xml version="1.0" encoding ="UTF-8"?>
* <callscript>
* <call>
* <command name="tel" type="hangup delay="10"/>
```

```
* </call>
* <call>
* <command name="tel" type="answered"/>
* </call>
* </callscript>
*
*
* Example of handling a transfer, hot word recognition.
*
* <?xml version="1.0" encoding ="UTF-8"?>
* <callscript>
* <call>
* <command name="rec" type="asr">
* <item value="dog blue"/>
* </command>
* </call>
* <call>
* <command name="tel" type="answered"/>
* </call>
* </callscript>
*
* Example of merging 2 calls.
*
* <?xml version="1.0" encoding="UTF-8"?>
* <callscript>
* <call>
* <command name="merge" value="okay"/>
* </call>
* <call>
* <command name="tel" type="answered"/>
* <command name="merge" value="okay"/>
* </call>
* </callscript>
*
* Note any command can have a delay attribute. The delay means that the
* command with the delay is activated delay seconds after the preceeding command is
* processed. In the far end disconnect example, the hangup is sent 10 seconds after the
* call is answered. In the near end disconnect the hangup is sent 10 seconds after the
* initial incoming call. Note that the first call in the script never has an answer
* that is implicit. If you do answer the incoming call, an error will result.
*/
```

Parameters Tab

The Parameters tab allows the application developer to configure VXML parameters to simulate different settings or configurations in VXML and CCXML applications, as if the application was being launched via a VXML or CCXML Web service on the Voice Portal.

Tab Area	Description and Usage
Call Control Protocol Parameters	Allows for protocol specific parameters to be set, and passed when simulating your application. Entered parameters must be the following format: **session.connection[<protocol_name>].*** Enter a protocol name and version. Then enter a parameter name and value, and click **Add Param**. In a real deployment, these values might otherwise be set automatically by the supporting platform (for example, Voice Portal platform)
Call Control Session Parameters	Allows for session-based parameters to be set, for a VXML session to simulate launching the CCXML Web services. Entered parameters must be the following format: **session.*** Enter a protocol name and version. Then enter a parameter name and value, and click **Add Param**. In a real deployment, these values might otherwise be set automatically by the supporting platform (for example, the Voice Portal platform)
AAI/UUI Options	These options allow you to set Voice Portal specific UUI/AAI parameters. There are two modes: • Shared - Means that the format of the configured value is fixed, and as specified by Voice Portal. • Service Provider - Means that the value in uninterpreted. If the mode is shared, the following considerations apply: • VXML - session.connection.aai is unpacked into the shareduui varaible; session.avaya.uui.mode stored in session:sharedmode; if there is a UCID in the AAI, it is decoded into session.avaya.ucid and session:sessionlabel is set to the UCID • CCXML - connection.aai is unpacked into connection.avaya.uui.shared; mode is in connection.avaya.uui.mode; if there is a UCID in the AAI, it is decoded into connection.avaya.ucid An example of the AAI is: **12, 01DR12345678; FA, 0001001C46E70028** where 12 and FA is the application ID, and the remaining characters in the string are the values, respectively. For additional details, see the Avaya Voice Portal documentation.

Tab Area	Description and Usage
Call Classification	These parameters allow you to simulate different types of calls at the voice platform in CCXML. The types of calls are: • live_voice - person answered • fax_calling_tone - sending tone, inbound call from a fax machine • timeout - timeout on classification expired • error The call classifications are passed to an application via the “call classification” query string variable to the Web Service.
Speech Application Parameters	These parameters, when set to values in a VXML session take the parameters in the namelist. Entered parameters must be the following format: **session.connection[<protocol_name>].*** Enter a protocol name and version. Then enter a parameter name and value, and click **Add Param**. The parameter is added to the table list, and passed to the simulated application. It allows you to test your application as though it was launched in CCXML. In a real deployment, these values might otherwise be set automatically by the supporting platform (for example, IR or the Voice Portal platform)

Problems View

The **Problems** view automatically displays any errors, warnings, or informational messages generated while saving a project or any of its elements. At the same time, Dialog Designer displays a Code Generation error message, unless you click **Do not show me this message in the future.** This view also shows any errors generated while compiling the Java-code.

In most cases, if the error, warning, or informational message entry is clicked in this view, Dialog Designer directs the focus to the exact location where the problem occurs. This feature makes it easier to debug an application and fix the problem.

By default, Dialog Designer displays the **Problems** view in a tabbed notebook, along with the **Avaya Application Simulator** view and the **Tasks** view, in the lower-center area of the window.

Tasks View

The **Tasks** view is a standard Eclipse view. It automatically displays tasks related to various types of errors that might occur, such as Java syntax errors. You can also manually add tasks to this list, for things you do not want to forget to take care of.

For more information about the **Tasks** view in Eclipse, see the topic “Tasks view” in the Eclipse *Workbench User Guide* online Help.

By default, Dialog Designer displays the **Tasks** view in a tabbed notebook, along with the **Avaya Application Simulator** view and the **Problems** view, in the lower-center area of the window.

Eclipse Properties View

The **Eclipse Properties** view is a modification of the standard Eclipse Properties view. In a layout optimized for Dialog Designer, it displays property names and values for nodes, palette options, or other items. The properties available for editing vary according to the editor, node, or item you are working with. For more information about the properties you can edit for a particular node, option, or other item, see the online Help topic for that node, option, or item.

For more information about the standard Properties view in Eclipse, see the topic “Properties view” in the Eclipse *Workbench User Guide* online Help.

By default, Dialog Designer displays the **Eclipse Properties** view in a tabbed notebook, along with the **Console** view, in the lower-right area of the window. Dialog Designer brings the **Eclipse Properties** view to the foreground whenever clicking on the **Eclipse Properties** tab.

Console View

The **Console** view displays information about the Tomcat server status and activity. If you have the debug output for tracing function turned on, this view also displays the VoiceXML output generated by the application. This information is read only, but it might be helpful in debugging applications, especially if you can read and understand VoiceXML code.

To turn on the debug output for tracing function:

1. In the **Window** menu, click **Preferences**.

   The system displays the Preferences window.

2. In the navigation pane on the left, click **Dialog Designer** > **Avaya Application Simulator**.
3. Among the options on the right side of the **Preferences** window, verify that the check box labeled **Enable Dialog Designer logging of tracing output** is selected.
4. Click **OK**.

Tip:

With this function turned on, Dialog Designer displays the VoiceXML output in the **Console** view and writes the output to a trace log file. This file is located at ***applicationName*/data/log/trace.log-*yyyy-mm-dd*.log**, where *applicationName* represents the top level application directory, and *yyyy-mm-dd* represents "today's" simulation runs generated. To view this log file, locate the file in the **Navigator** pane and double-click the file name.

By default, Dialog Designer displays the **Console** view in a tabbed notebook, along with the **Properties** view, in the lower-right area of the window. The **Console** view is brought to the foreground whenever you start Tomcat or click the **Console** tab.

Dialog Designer Menu and Toolbar Options

There are several main menu and toolbar options that are specific to Dialog Designer. The following table presents a quick reference and summary of these options.

Note:
This table presents only the options that are specific to Dialog Designer. Generic Eclipse options are not included.

Dialog Designer Menu and Toolbar Options

Icon	Descriptor	Function/Comments	Where Available
	Speech Project	*Speech Perspective only*. Opens the wizard to create a speech project.	• **File** > **New** menu • Main toolbar • **Navigator** context (right-click) menu
	CCXML File Editor	*Call Control Perspective only*. Opens an editor to edit CCXML files.	• **File** > **New** menu • Main toolbar • **Navigator** context (right-click) menu
	JSP File Editor	*Call Control Perspective only*. Opens an editor to edit JSP files.	• **File** > **New** menu • Main toolbar • **Navigator** context (right-click) menu
	Call Control Project	*Call Control Perspective only*. Opens a wizard to create a call control project.	• **File** > **New** menu • Main toolbar • **Navigator** context (right-click) menu
	Database Operation File	*Call Control or Speech Perspective*. Opens the wizard to create a database operation file (can be used in either a speech application or call control application).	• **File** > **New** menu • Main toolbar • **Navigator** context (right-click) menu
	Grammar File	*Speech Perspective only*. Opens the wizard to create a grammar file.	• **File** > **New** menu • Main toolbar • **Navigator** context (right-click) menu
	Phraseset file	*Speech Perspective only*. Opens the wizard to create a phraseset file.	• **File** > **New** menu • Main toolbar • **Navigator** context (right-click) menu
	Prompt File	*Speech Perspective only*. Opens the wizard to create a prompt file.	• **File** > **New** menu • Main toolbar • **Navigator** context (right-click) menu

Dialog Designer Menu and Toolbar Options (continued)

Icon	Descriptor	Function/Comments	Where Available
	Web Service Operation File	*Call Control or Speech Perspective*. Opens the wizard to create a Web service operation file (can be used in either a speech application or call control application).	• **File** > **New** menu • Main toolbar • **Navigator** context (right-click) menu
	Start Tomcat	*Call Control or Speech Perspective*. Starts the Tomcat servlet engine, which is required to simulate applications in Dialog Designer.	• **Tomcat** menu • Main toolbar
	Stop Tomcat	*Call Control or Speech Perspective*. Stops the Tomcat servlet engine, which is required to simulate applications in Dialog Designer.	• **Tomcat** menu • Main toolbar
	Restart Tomcat	*Call Control or Speech Perspective*. Restarts the Tomcat servlet engine, which is required to simulate applications in Dialog Designer.	• **Tomcat** menu • Main toolbar
No icon	**Update Context** option	*Call Control or Speech Perspective*. Updates the Tomcat server with the current application information, known as the context definition. This action is sometimes required when you update the application but changes do not get communicated to the Tomcat server. One indication that this might be required is if you get a “General Error 404" message.	• Right-click on **Project** in Navigator, and select **Dialog Designer > Update Context**.
No icon	**Generate** option	*Call Control or Speech Perspective*. Generates the project code.	• Right-click on **Project** in Navigator, and select **Dialog Designer > Generate**.

Dialog Designer Menu and Toolbar Options (continued)

Icon	Descriptor	Function/Comments	Where Available
No icon	**Validate** option	*Call Control or Speech Perspective*. Validates the Call Flow.	• Right-click on **Project** in Navigator, and select **Dialog Designer > Generate**. • Expand project in Navigator • Right-click on **flow/ main.flow** file, and select **Dialog Designer > Validate**.
	Event Type Editor	Opens the Event Type Editor.	Main toolbar, Call Flow Editor only
	Add row after	Adds a row to a grammar table, inserting the new row *after* the selected row.	Main toolbar, Grammar File Editor only
	Add row before	Adds a row to a grammar table, inserting the new row *before* the selected row.	Main toolbar, Grammar File Editor only
	Add column after	Adds a column to a grammar table, inserting the new column *after* the selected column.	Main toolbar, Grammar File Editor only
	Add column before	Adds a column to a grammar table, inserting the new column *before* the selected column.	Main toolbar, Grammar File Editor only
	Delete row	Deletes the selected row from a grammar table.	Main toolbar, Grammar File Editor only
	Delete column	Deletes the selected column from a grammar table.	Main toolbar, Grammar File Editor only

Chapter 4: Creating Speech Applications with Dialog Designer

Dialog Designer greatly simplifies the process of creating VoiceXML-compliant speech applications for use on Avaya Interactive Response and Voice Portal systems.

This chapter is designed to help you better understand the process and provide practical “best practices” ideas for creating your own speech applications. It also provides a checklist that is useful to use during the application design and development process.

It contains the following sections:

Planning Before You Build

The importance of careful planning and design *before* starting to actually build a speech application cannot be emphasized enough. Investing the time in planning the application at the start of the process can greatly reduce development time. This investment can also help you avoid incurring problems later on.

Review the following important aspects when planning an application:

Envision the User Experience

To start planning a speech application, envision exactly what you want the caller to experience when calling into your system. Without being concerned with how to set it up, simply ask and attempt to answer as many questions as possible. For example:

- 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?
- What scenarios or situations do you think callers might require help with?

These questions are just a sampling. Try to be as comprehensive as possible.

Anticipate Potential Problems and Issues

Another important area in planning a good speech application is to plan for any potential problems and error conditions that you can think of in advance. How is the system to respond when one of these problem situations arises?

- **What are the technical or hardware limitations?** What if the caller does not have a touchtone telephone? How does your system respond to TTY or TDD requests?
- **Is accessibility for callers with physical limitations required?** Have you allowed for the extra time it might take for callers with physical handicaps or other limitations?
- **What language limitations will be incurred?** Is it likely that some of your callers will need to interact using a different language than the primary language? Do you have the necessary ASR and TTS servers and software to accommodate them?

- **What personal preferences are needed for callers?** Have you allowed for the personal preferences of your callers? Some people would rather interact with the system verbally, while others might prefer to use DTMF (touchtone) responses. And, some people prefer to interact with a live attendant no matter *how* good your interactive voice response (IVR) speech application is.

  Such preferences are among the reasons that it is recommended that you *always* provide callers with an option to speak with a live attendant. Beyond personal preferences, it is likely that your system cannot deal with all eventualities and situations a caller might need help with. There are times when only a live attendant can take care of the need.

Map the Flow

While envisioning the experience that you want your callers to have, try to foresee and plan for any problem contingencies that could arise. Begin to map the flow of the speech application. Following are a couple of the common approaches to help get started:

- **Describe the flow verbally.** Verbally talk through each of scenario. Take note of where the prompts occur and what callers should say or do. Record these verbal “walk-throughs.”
- **Use a flow diagram.** Create a flow diagram to show the major points in the call flow. Use this diagram to show:
  - Where to offer options to callers
  - Where callers should listen to the entire prompt and where they can interrupt, or “barge in,” and cut the off prompt
  - Where a response from callers is required and what the valid responses should be
  - Where the application will need go to access databases for retrieving or recording customer data
  - How and where the application needs to access a Web service to respond to a customer request

    And so on...

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

Make It Modular

When planning a speech application, be alert to places where parts of the application can be reused in more than one place. Subsequently, when designing and building the application, create these parts as speech project modules that can reused wherever that functionality is needed.

If these modules are planned ahead of time, they can be developed before developing the main application project file. This way, they are already available when creating your main application.

For example, if bank account or credit card numbers need to be collected from callers at several points in the call flow, the same basic process to collect such numbers, regardless of the types of numbers, may be the same. Therefore, a reusable sub module in the master application might be the process of collecting account numbers from callers.

Advantages of Modular Design

Using a modular approach to application design has several advantages:

- **Develop once, use many times.**There is a a tremendous efficiency advantage to have certain actions or options to be offered to callers in several places within the application.
- **Maintenance ease.** It is easier to maintain the application, even if you are not reusing much of the code. When using a modular approach, code can be changed in one part of the application without necessarily requiring a rebuild of the entire application.
- **Debugging ease.** It can make it easier to debug an application by being able to isolate the trouble areas where errors are occurring.
- **Concurrent engineering capabilities.** A team of developers can work on separate pieces of an application separately and then merge their efforts.

Process to Create Modules

The process of creating modules for reuse is a three step process, as follows:

1. Use Dialog Designer to create a speech project that contains the functionality for the module.
2. Export the speech or call control project to the correct directory for reusable dialogs.
3. Import the speech or call control project module into another speech or call control project.

See Deploying Projects as Dialog Designer Modules.

Identify and List Application Resources

While planning a speech application, it is recommended that all required application resources be identified and listed before developing the application. For a list and description of the types of application resources available in Dialog Designer, see Using Application Resources.

If you have planned the flow completely, keeping in mind all the required resources, these application resources can be listed before actually starting to develop the application. Then, relevant resources can be created and imported before creating the call flow.

In the case of phrases, for example, if you know exactly what phrases will be needed, 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 starting to develop the actual application. Then, when developing the application, when you get to the points in the application where you need the phrases, you can import the prerecorded files quickly and easily.

Naming Java Components

When deciding on names for Java components such as projects, folders, and files, the following conventions and restrictions apply.

Java component names:

- Cannot contain spaces
- Are case-sensitive
- Can use, for the initial character:
  - An alphabetic character, either lowercase or uppercase
  - An underscore character (_)
- Can use, for subsequent characters:
  - Alphabetic characters, either lowercase or uppercase
  - Underscore characters (_)
  - Numbers

CAUTION:
Do not use double-byte or extended ASCII characters when naming Java components.

Using the Speech Project Wizard

Once you have planned an application (see Planning Before You Build), you are ready to use Dialog Designer to create the actual application. Use the Speech Project Wizard to create the basic framework for the speech application project.

The following topics explain how to use the Speech Project Wizard:

- Accessing the Speech Project Wizard
- New Speech Project Page (Required)
- Project Options Page (Optional)

- Specify Language Parameters Page (Optional)

Accessing the Speech Project Wizard

To access the Speech Project Wizard, use one of the following methodologies:

- From the **File** menu, select **New** > **Speech Project**.
- On the main toolbar, click the Speech Project icon.
- Right-click anywhere in the **Navigator** view, then select **New** > **Speech Project**.

After performing any of these actions, Dialog Designer displays the New Speech Project page.

New Speech Project Page (Required)

Use the **New Speech Project** page to name the project and set the location where Dialog Designer is to save the project.

- **Project Name:** field - Type in the name to assign to the project.

  CAUTION:
  Do not use double-byte or extended ASCII characters when naming the project.

  Note:
  Project names must follow Java naming conventions (see Naming Java Components).

- **Project contents** options:
  - To use the default project location for generating and saving files (recommended), select the **Use default** check box.
  - To use a different location, clear the **Use default** check box and perform one of the following actions:
    - In the **Directory** field, type the full path of the directory in which you want to build and save project files.
    - Click **Browse** to locate the directory in which you want to build and save project files.

After setting the project name and directory location:

- Click **Next** to set other project options. See:
  - Project Options Page (Optional)
  - Specify Language Parameters Page (Optional)

- To create the basic project without setting other options, click **Finish**.

  Clicking **Finish** opens the Call Flow Editor, where you design and build your project call flow.

  See Working with the Call Flow Editor.

Project Options Page (Optional)

Use the **Project Options** page to record “meta information” and a description for your project. Dialog Designer does not use this information in the application itself, nor does it appear anywhere in the output. You can consider this information to be primarily internal information. Dialog Designer saves and displays this information only as part of the project properties. However, it can still be helpful to use these fields.

For example, if you are developing similar applications or slight variations of an application for several different clients, you might use these fields to indicate who the clients are and what the variations consist of.

The fields on this page include:

- **Project** options (**Mode** and **Type**) - Unavailable in the current release of Dialog Designer
- **Meta Information** options
  - **Vendor:** - Type in your name or the name of your company.
  - **Category:** - Type in a short descriptor that tells what type of application or project module this is.

    For example, if you are building a banking application and this module is being developed to handle savings account queries, you might type in something like: **Savings query module**.

    Tip:
    If this project is to be used as a reusable module, what is entered in this field determines where in the Call Flow Editor palette this module will appear. If this field is left blank, the module appears in a group labeled “Modules.” With an entry in this field, the module is grouped with other modules that have the same category label.
- **Description** option - Type in a brief description of what you are designing this project to do.

  Avaya recommends that you include information like the variable values that are accepted and returned from this module and other similar information.

To view or modify this information later, access the project properties dialog box. See Setting Project Properties.

After setting options on this page:

- Click **Next** to set the primary language options for the project (optional).

  See Specify Language Parameters Page (Optional).

- Click **Finish** to create the basic project without setting other options. Clicking **Finish** opens the Call Flow Editor, where you design and build your project call flow.

  See Working with the Call Flow Editor.

Specify Language Parameters Page (Optional)

Use the **Specify Language Parameters** page to set the options for the primary language to be used in your project. Note that you can only select one (primary) language for each language option here. If you need to add more languages, you can do so later. See Language and Localization Considerations.

Parameter	Description
Language Package Name	Type in a logical name to assign to the primary language for your project. The name does not impact the runtime language settings. This is the name that Dialog Designer uses for the language directory in the **Navigator** view. **Tip:** Avaya recommends that you use a descriptive name, such as **US_Eng** for U.S. English or **Can_Fr** for Canadian French. For file-naming conventions and restrictions, see Naming Java Components.
Speech Recognition (ASR) Language	Click the arrow, then click the name of the ASR package to use for speech recognition. This field is populated with the list of ASR languages supported in Dialog Designer. **Note**: Check the documentation of your ASR runtime engine and use of the engine-supported language codes configured on the voice platform. For example, some engines/platforms will not accept “en-us” and instead required “en-US”.

Parameter	Description
Text-to-Speech (TTS) Language	Click the arrow, then click the name of the TTS package to use for speech synthesis. This field is populated with the list of TTS languages supported in Dialog Designer. **Note**: Check the documentation of your TTS runtime engine and use of the engine-supported language codes configured on the voice platform. For example, some engines/platforms will not accept “en-us” and instead required “en-US”.
Localization Bundle Language	Click the arrow, then click the name of the localization bundle to use. Localization bundles are packages of classes in a *.jar file that the servlet engine uses to convert run-time application data to the form of audio variables. The VoiceXML interpreter, Avaya Application Simulator, translates this run-time data to standard phrases and plays the phrases back in the form of prerecorded audio files. Dialog Designer populates this field with the list of localization bundles supported in Dialog Designer.

Building Applications

The basic process of building speech application projects in Dialog Designer involves:

- Placing nodes in the workspace area
- Connecting them to create a call flow

This call flow represents the possible paths a caller can follow while interacting with your interactive voice response (IVR) system. See Working with the Call Flow Editor.

About Nodes

Each *node* is represented by an icon and label on the options palette. When placed in the workspace, the node represents a functional piece of code that performs some action, such as:

- An announcement that is played to callers when they first call in to the system.
- A menu that offers callers options from which they can select, responding either by speaking or by pressing touchtone keys on their telephones.
- A data transaction that sends data collected from a caller to a customer database. The data transaction then retrieves information from that database to present to the caller.

For more information on the types of nodes available in Dialog Designer, see Getting Familiar with Nodes to Build Applications.

Placing Nodes and Other Items

Dialog Designer requires a click-and-drop technique to place nodes and other items in the Editor view workspace. Usually, to place a node or item in the workspace:

1. Click the desired node or item on the options palette.
2. Click again in the workspace at the location you want to place the node or item.

Specific details for placing nodes or items in the workspace are provided, if applicable, where those nodes or items are discussed.

Connecting Nodes

Once two or more nodes are placed in the workspace, the can be connected according to the call flow design. To connect nodes, see Working with the Call Flow Editor.

Defining Nodes and Other Items

Once the system displays the node or item in the workspace, the node can be further defined or modified. Do this by clicking the node icon and then filling in or selecting properties in the **Properties** view. Some properties can be directly edited in the node editor for that node.

Almost any node or item can be edited in the Editor view workspace. In fact, to make sure the work the way they are intended to work, they *must* be edited. For more information about editing nodes and other flow items, see Call Flow Editor Palette Action Options and Nodes and Palette Options.

Many nodes also require being populated with one or more application resources, such as phrases, prompts, and grammars. See Using Application Resources.

Using Application Resources

Many nodes and other flow items in Dialog Designer make use of various application resources, such as phrases, prompts, and grammars. These application resources are often created using their own wizards, and they are defined or modified in their own editors.

For example, before the Announce node is fully functional, Dialog Designer requires that:

- The Announce node be populated with a prompt.
- Where the call flow is to go after completing the Announce node be defined
- One or more prompts be created using the **Prompt File wizard**.
- The prompt(s) be assigned to the Announce node.

Tip:

If you have carefully planned your application, and you know ahead of time what application resources will be needed, Avaya recommends that you go ahead and create or import those application resources before actually starting to build your call flow. That way, they are ready and available when needed.

Following is a list of the application resources available in Dialog Designer:

- **Phrases** are prerecorded audio files that are called and played back to callers or text strings that are synthesized into audible form using Text-to-Speech (TTS) technology.

  See About Phrases.

- **Phrasesets** are used to group phrases, usually related to a particular speech application. The main advantage for grouping phrases, aside from better organization, is that with phrasesets, file resource efficiencies are greatly improved which translates into quicker build times.

  See About Phrasesets.

- **Prompts** are speech elements used to announce information to a caller or prompt the caller for some type of response. They are composed of one or more segments. These segments can consist of phrases, audio variables, text variables, or TTS text. You can also use conditional statements to determine which prompt segments are used in certain circumstances.

  See Working with Prompts.

- **Variables** in Dialog Designer can take any of several different specialized forms. Some of the specialized forms include audio variables and text variables used for TTS, and the standard programming variable types.

  See Working with Variables.

- **Grammars** are speech elements used in conjunction with automated speech recognition (ASR) technology. Grammars are lists of possible responses that callers might make when they respond to prompts using spoken replies. Grammars define which words or phrases the ASR engine can recognize and respond to.

  For more informations, see Working with Grammars.

- **Languages** - In Dialog Designer applications, you can assign more than one language to a speech project. You can create applications that offer callers the option to select languages, as long as those languages are available on your system.

  See Language and Localization Considerations.

- **Database operations** - Use database operations to connect and interact with SQL databases. Using database operations, you can take information collected from a caller and write that information to the database. You can also retrieve information from the database to present the information to the caller.

  See Working with Database Operations.

- **Web service operations** - Web services are Internet-based applications that you can use to perform a wide variety of functions. In Dialog Designer, you can invoke Web services with a Web service operation file.

  For example, you can use a Web service to enable callers to get stock quotes or weather forecasts for their area. Or you can use Web services to enable callers to find out about the availability of airline flights or hotel reservations. Nearly any Web service in which the information can be presented using audio files or Text-to-Speech can be used in a Dialog Designer application.

  See Working with Web Services.

- **Event types** - Dialog Designer includes most of the common types of event handlers as part of the options available from the palette. In addition, you can use the Event Handler Editor to create your own custom event types.

  For example, you can create a custom event handler to throw an exception whenever the amount of a transaction request exceeds the currently available limit on a credit card account belonging to the caller. Then, you can place this event type in the application so that, when the application throws this event, the application goes to a node that informs the caller that the credit limit does not allow this transaction to complete.

  See Working with Event Types.

Cut, Copy, and Paste Operations

Dialog Designer supports copy and paste operations for elements of an expanded node (such as logic, use of resources, etc.) from one project to another. Dialog Designer also supports copying, cutting, and pasting a set of nodes and supporting resources, or a set of resources including prompts, phrases and grammar, in the following scenarios:

- From one project to another project
- From within a project to another part of a project within the same call flow

Upon using the copy, cut, and/or paste operations, all names are preserved unless conflicts are discovered, in which case modified similar names are auto-generated.

To invoke a copy or cut (and eventually paste) operations (in the Call Flow Editor), select a node, resource, prompt, phrase, or grammar, then right-click to select **Copy** or **Cut**. The selected object is copied to the Dialog Designer clipboard.

To paste the clipboard object, similarly, click in the location where the object should be pasted, then right-click to select **Paste**. The copied object should appear, with the same or modified name, depending on whether the operation was performed in the same project or between different projects, respectively.

Caveats

Before using the cut, copy, and paste operations within Dialog Designer, Avaya recommends reviewing the following caveat statements to better understand the functionality and limitations:

- When cutting from a call flow, nodes or items from the source call flow are deleted. Any variables that are defined by the call flow are also deleted.
- When copying across projects, the entire resource dependency tree is copied to the target project (prompts, grammars, phrases, variables, and so on).
- When the resource dependency tree is copied, certain variables may not be copied because they are implicitly created by other project elements (for example, a Prompt & Collect node). So if the project element is not part of the clipboard contents, then the variable may not be copied to the target project.
- When copying across projects with multiple languages, Dialog Designer attempts to determine where language resources belong upon pasting. If there are different languages in the source/target project, after-paste manual steps may be required to get things aligned properly.
- To undo a copy and paste operation, use ctrl-Z (however, once a Save has been applied, a copy and paste cannot be undone).
- When splitting large callflows into multiple callflows, cut should be used to move nodes from one flow to the other rather than copy. Copy will result in creating variables, etc. Cut will move the nodes from one flow to the other.

Testing Applications

Before you deploy your application, and often during the application development process, Avaya recommends that you test your application. Dialog Designer offers an application simulation and debugging environment in which you can exhaustively test your application before “going live” with the application. Using the simulation and debugging tools, you can verify that the application works the way you want. You can also eliminate most, if not all, problems with your application before you ever deploy the application to a live system.

While testing an application by simulation, you can:

- Specify input parameters
- Simulate a variety of error conditions
- Test for DTMF and spoken responses
- Interact with databases, in actuality or by simulation
- Invoke Web services

See Application Testing by Simulation.

Deploying Applications

The final step in developing a speech application project is to deploy the application, so that the application can be used as intended. How you deploy the project depends on what you intend the project to do.

Before deploying the project, you must build the project and generate the code. Depending on what application execution environment you are creating the application for, the deployment process then creates a Web archive (WAR) file or enterprise archive (EAR) file.

If you designed the project to be a sub-dialog or reusable module for a larger application, you must select an option that deploys the project to a directory for reusable modules. You can then import that module into another speech project and use the project as a module node.

If you designed the project to be the primary application or a stand-alone project, you can transport the WAR or EAR files to the server on which you designed the application to execute.

For more information on deploying applications, see the following sections:

- Module Nodes
- Application Deployment

Checklist for Developing Applications with Dialog Designer

The following checklist reflects the recommended approach to designing and building speech application projects with Dialog Designer.

Use the checklist to help guide you through the speech application development process.

Process step:	Reference
Plan the application: 1. Envision the user experience. 2. Foresee potential problems. 3. Plan the flow. 4. Plan for any modules to be reused. 5. List any application resources that must be created or imported.	Planning Before You Build Using Application Resources
Create the speech application project.	Using the Speech Project Wizard

Process step:	Reference
Create or import the required application resources.	Using Application Resources
Build the application: 1. Place applicable nodes in the Call Flow Editor workspace. 2. Connect the nodes as required. 3. Edit the nodes as required.	Building Applications Call Flow Editor Palette Action Options
Test the application before deploying.	Testing Applications
Deploy the application to its run-time location.	Deploying Applications

Chapter 5: Creating Call Control Applications with Dialog Designer

Dialog Designer greatly simplifies the process of creating CCXML call control applications for use on Voice Portal systems.

This chapter is designed to help you better understand the process and provide practical “best practices” ideas for creating your own call control applications. It also provides a checklist that is useful during application design and development.

It contains the following sections:

About CCXML, Dialog Designer, and Call Control Applications

Dialog Designer, with Release 4.0, supports the creation of call control applications based on the standard language for managing telephony call control state-based applications—CCXML (Call Control eXtensible Markup Language). A call control CCXML application is a collection of CCXML and JSP pages that manage call control objects, such as connections, dialogs, CCXML sessions, etc.

Note:
Unsupported features of CCXML in this release of Dialog Designer for Voice Portal or SImulation include: Conferencing (createconference(), destroyconference(), merge(), move()), spawning of a new CCXML session from an existing session (createccxml()), and Basic HTTP I/O processor operations.

Dialog Designer’s support of CCXML provides a lower level of support than do VXML speech applications, also created in Dialog Designer. By simple standard definition, CCXML provides a more detail oriented and finer grained control in caller application development, when needed.

The Dialog Designer 4.0 implementation is based on the June 29, 2005 version of the CCXML standard, accessible at: http://www.w3.org/TR/2005/WD-ccxml-20050629/

The purpose of this guide is to provide direction and guidance on working and implementing CCXML while developing your Dialog Designer CCXML call control applications, not to teach you CCXML itself. To learn CCXML, read the specification at the referenced link, and study the available sample call control applications included with Dialog Designer.

To evaluate, review, and test existing call control sample applications in Dialog Designer, open the Call Control Perspective (**Window > Open Perspective > Call Control**). This Eclipse perspective is similar to the Speech Perspective in look and feel, but it has been created specifically for call control projects, and optimized for text-based editing of CCXML and JSP pages—the crux of call control applications.

In addition to many of the same tabs found in the Speech Perspective, one main unique one to the Call Control Perspective is the Snippets tab. See Building Call Control Applications.

Using the Call Control Project Wizard

Once you have planned a call control application, you are ready to use Dialog Designer to create the actual application. Call Control projects are created via the Eclipse Call Control Project Wizard. The Call Control wizard walks you through the creation of a new Call Control project to create the basic Call Control project structure and a default CCXML start page (using a CCXML template).

Use the Call Control Project Wizard to create the basic framework for the call control application project. Call control projects can be made up of VXML, CCXML, and JSP pages.

The following topics explain how to use the Call Control Project Wizard:

Accessing the Call Control Project Wizard

To access the Call Control Project Wizard, use one of the following methodologies:

- From the **File** menu, select **New** > **Call Control Project**.
- On the main toolbar, click the Call Control Project icon.
- Right-click anywhere in the **Navigator** view, then select **New** > **Call Control Project**.

After performing any of these actions, Dialog Designer displays the New Call Control Project page.

New Call Control Project Page (Required)

Use the **New Call Control Project** page to name the project and set the location where Dialog Designer is to save the project.

- **Project Name** – Type in the name to assign to the project.

  CAUTION:
  Do not use double-byte or extended ASCII characters when naming the project.

  Note:
  Project names must follow Java naming conventions.

- **Location** – By default, the currently set Dialog Designer workspace path and directory are specified for where the new call control project will be saved. This is set with a “checked” **Use default location** flag.

  To specify a different location, unselect the flag, and use the **Browse** button to navigate to a different directory.

To continue to define your call control project, click either of the following options:

- Click **Next** to set other project parameters. See Specify Project Parameters Page (Optional).
- Click **Finish** to create a basic call control project “container” without setting other parameters. Clicking **Finish** opens the Call Flow Editor, where you design and build your project call flow. See Working with the Call Flow Editor.

Specify Project Parameters Page (Optional)

Use the **Project Options** page to record “meta information” and a description for your project. Dialog Designer does not use this information in the application itself, nor does it appear anywhere in the output. You can consider this information to be primarily internal information. Dialog Designer saves and displays this information only as part of the project properties. However, it can still be helpful to use these fields.

For example, if you are developing similar applications or slight variations of an application for several different clients, you might use these fields to indicate who the clients are and what the variations consist of.

Call Control Project Wizard—Specify Project Parameters Page

Field	Description
Project	
Mode	Unavailable in the current release of Dialog Designer.
Type	Unavailable in the current release of Dialog Designer.
Meta Information	
Vendor	Name or the name of your company.
Category	Short descriptor of the type of application or project module.
Description	Brief description of what the project is designed to do.

Click **Next** to select a CCXML template. See Select CCXML Template Page (Optional).

After setting options on this page, click **Finish,** to create the basic project without setting other options.

Select CCXML Template Page (Optional)

Use the **Select CCXML Template** page to get started building and defining your call control project using a CCXML template, a CCXML page with initial content for a default start page.

Templates allow you to get a jump on CCXML content (and JSP content, if using a JSP template) and elements that will be used in you CCXML pages, and ultimately your call control application. The context of the template is used to determine where the CCXML will show up.

Once defined, CCXML templates are also accessible, reviewable, and editable through the Dialog Designer CCXML Templates Preferences page (see CCXML Templates). Upon opening the CCXML Template Wizard (also available by clicking on the CCXML Wizard icon on the main toolbar within the Call Control Wizard), the developer is presented with two templates to begin building out their CCXML:

- Basic – Contains basic CCXML content and structure.
- Accept call – Contains basic CCXML content with transitions to accept a call.

Alternatively, a JSP page can be created for the call control project using the JSP FIle Wizard. See Using the JSP Wizard.

Building Call Control Applications

After creating the base call control project using the Call Control Project Wizard, the base components of the new call control project appears in the Navigator panel with a red colored Dialog Designer project icon. Call control projects have a file structure to organize the contents of your project.

- A *ccxml* folder used to store your CCXML pages including the default **start.ccxml** page.
- A *jsp* folder used to store your JSP pages.
- A *vxml* folder used to store static VXML pages.
- A *connectivity* folder used for database and Web service operations.
- Other similar folders and files that are also common with speech projects.

The preview pane—the CCXML editor—shows the initial content to be edited for your call control project. The basic process of building call control application projects in Dialog Designer involves:

- Creating, editing, and evolving CCXML templates in the CCXML Editor.
- Creating, editing, and evolving JSP pages, as needed in your application.

Dialog Designer leverages the Eclipse WebTools Platform for content aware editors. The CCXML Editor is essentially an XML editor. In the same, you may adjust the look and feel of how your code is displayed. (Go to **Window > Preferences > Web and XML > XML Files > XML Styles** or **Window > Preferences > Web and XML > JSP Files > JSP Templates**, for example.)

See the following sections:

- Working in the CCXML Editor
- Using Database and Web Service Operations
- Using the JSP Wizard

Working in the CCXML Editor

The CCXML Editor is used for editing CCXML and JSP templates and pages to create the logic of your call control project. Templates created in a template editor show up based on your cursor position to allow you to enter your own content.

After first creating a base call control project using the Call Control Project Wizard (see Using the Call Control Project Wizard), the Eclipse display of your base call control project shows numerous different tabs to assist in editing and evolving your call control application.

- **Problems** tab (lower center area tab) – If any illegal content, these “problems” show up as a warning or error.

- **Outline** tab (upper right column tab) – Uses a hierarchical tree view to display the CCXML code. Content can be edited within tab.
- **Properties** tab (lower right column tab) – Uses a table view to display the CCXML code. Content can be edited within tab.
- **Design** tab (lower left tab in main window area) – Uses a hierarchical tree view. Content can be edited within tab.
- **Snippets** tab (lower center area tab) – Contains clickable “drawers” of JSP or CCXML code snippets that can be added to CCXML or JSP pages. Add snippets to your CCXML code in one of the following ways:
  - Drag the snippet to the location where the snippet should be added in the CCXML page and release. The snippet is added to your code.
  - Put the cursor in the location where the snippet should be added in the CCXML page. Click on the snippet type, and the snippet is added to your code.

  Depending on the snippet type, in some cases a subsequent dialog appears to allow for the entry of variable data to complete the snippet before it is added to your code.

  Since call control projects have no variables, you map only to a variable name of your choosing. The variable will be created on the fly in your script code.

  The Snippets tab provides easy to use CCXML fragments and content to invoke Database operations and Web service operations via proxies. See Using Database and Web Service Operations.

Avaya Application Simulator tab – Allows you to test your call control application in a simulated environment.

The type of content that can be edited/added is controlled by the **ccxml.dtd** that governs allowable content supported by the Voice Portal platform. This file can be found in the following directory: Eclipse/plugins/com.avaya.cc.core/dtd.

Important:
Not all CCXML commands from the CCXML specification are supported. To understand the supported CCXML command set, see the **ccxml.dtd** file.

Using Database and Web Service Operations

Database operations and Web Service operations in a call control project are used in a very similar way to how they are used in speech applications. Also, they are defined in the same manner using the same wizards.

Database operations are used to connect and interact with SQL databases. Information can be collected from a caller and then written to the database. Information can also be retrieved from the database to be presented to the caller. See Working with Database Operations.

Web services are Internet-based queries and applications that can be used to perform a wide variety of functions. In Dialog Designer, Web services are invoked with a Web service operation file.

An example Web service might be to enable callers to get stock quotes or weather forecasts for their area. Or a Web service to enable callers to find out about the availability of airline flights or hotel reservations. Nearly any Web service where information can be presented using audio files or Text-to-Speech can be used in a Dialog Designer application. See Working with Web Services.

At runtime, Dialog Designer will use Dbproxy and Wsproxy proxy classes to invoke Database operations and Web Service operations from within your CCXML content, respectively. Code to invoke Database operations or Web Service operations can be written by hand, or with call control projects. It is recommended to use the available Database operations and/or Web Service operations snippets on the Snippets tab within the CCXML Editor to automatically invoke these proxy classes for you.

The following transition invokes a DBOP by using the DbProxy in a fetch command. An input variable called ‘id’ is created, assigned and then passed in on the namelist as well as the ‘classname’ of the Database operation to invoke.

```
<transition event="connection.connected" name="evt" state="init">
    <log expr="'-- ' + evt.name +' -- [' + state +']'"/>
        <log expr="'   eventdata... \n' + objectToString(evt)"/>
        <assign name="state" expr="'invokeDB'"/>
        <var name="classname" expr="'connectivity.db.operations.getSayings'"/>
        <var name="id" expr="getRandomNumber(10)"/>
        <fetch next="'http://localhost:8080/InvokeDatabase/DbProxy'" type="'text/
ecmascript'"  namelist="id classname"/>
    </transition>
```

After the fetch completes, run the Database operation as follows:

Create a variable to obtain the output of the Database operation and results of the invocation. The <script> command invokes the script, runs the Database operation, and sets the output variables as well as the dbResult. In this example, we then use a “Saying” from the database to prepare a prompt to the caller.

```
<transition event="fetch.done" name="evt" state="invokeDB">
<log expr="'--In Event FETCH.DONE--'"/>
        <var name="Saying" expr="''"/>
        <var name="dbResult" expr="'OKAY'"/>
        <script fetchid="evt.fetchid"/>
        <if cond="dbResult == 'OKAY'" >
            <assign name="state" expr="'response'"/>
            <log expr="'Saying is:' + Saying"/>
           <dialogprepare type="'application/voicexml+xml'" connectionid =
"in_connectionid“ src="'http://localhost:8080/InvokeDatabase/vxml/Saying.vxml'"
namelist="Saying" />
       <else/>
             <log expr="'--Error in DB call:' + dbResult"/>
              <exit expr="'DB error'"/>
        </if>
    </transition>
```

For an array, you have to pass in the index:

```
<log expr="'Saying is : ' + Saying"/>
<log expr="'Saying is : ' + Saying[0]"/>

 namelist="Saying" />
 namelist="Saying[0]" />
```

Using the JSP Wizard

The JSP Wizard allows a developer to create JSP code using base JSP file templates, or additionally created JSP template files that include reusable blocks of code.

Templates allow you to get a jump on JSP content and elements that will be used in your CCXML code, and ultimately the call control application. The context of the template is used to determine where the CCXML will show up.

Once defined, JSP templates are also accessible, reviewable, and editable through the **Web and XML > JSP Files > JSP Templates** Preferences page.

Upon opening the JSP Template Wizard (by selecting **File > New > JSP File**, or clicking on the JSP Wizard icon ), the developer is presented with five templates to begin building out a JSP page (two are Dialog Designer JSP templates, and three are Eclipse templates):

- Basic – Dialog Designer template that contains basic JSP/CCXML content and structure.
- Accept call – Dialog Designer template that contains basic JSP/CCXML content with transitions to accept a call.
- New JSP File – Eclipse JSP template with HTML markup.
- New JSP File – Eclipse JSP template with xHTML markup.
- New JSP FIle – Eclipse JSP template with xHTML markup and XML style syntax.

By clicking on the *JSP Templates* link within the initial Wizard window, the JSP Preferences page is displayed (**Web and XML > JSP Files > JSP Templates**), showing additional JSP tag code segments that can be used (also provided by Eclipse).

Upon selecting one of the JSP template files, then clicking **Finish** in the Wizard, the code from the template file is added to a JSP project file within Eclipse. The developer can modify and save the JSP file as needed.

Deploying Call Control Applications

The final step in developing a call control application project is to deploy the application. How you deploy the project depends on what you intend the project to do.

Depending on what application execution environment you are creating the application for, the deployment process then creates a Web archive (WAR) file or enterprise archive (EAR) file. You can then transport the WAR or EAR files to the server on which you designed the application to execute.

For more information on deploying applications, see the following sections:

- Module Nodes
- Application Deployment

Chapter 6: Working with the Call Flow Editor

The Call Flow Editor provides the primary workspace and options for creating call flows in speech application projects. It also provides the means for setting global application properties and editing application flows in a node editor.

This chapter can help you understand the Call Flow Editor and how to use the Call Flow Editor to create call flows, set global properties, and edit nodes. It contains the following sections:

Note:
This section assumes that the user is already familiar with Eclipse concepts and terminology. If not, Avaya recommends reviewing Eclipse concepts and terminology now. See Accessing the Eclipse Workbench User Guide, “Concepts” Section.

Accessing the Call Flow Editor

To access the Call Flow Editor, perform any of the following actions:

- Create a speech project. When the Speech Project Wizard finishes, Dialog Designer automatically opens the Call Flow Editor. See Using the Speech Project Wizard.
- In the Editor view, click the tab labeled ***projectName*** **[main.flow]**, where *projectName* is the name you gave the speech project in the Speech Project Wizard. Use this action when multiple editors are open.

- Double-click the **main.flow** entry in the **Navigator** view. To locate this entry, click ***projectName*** > **flow** > **main.flow**. *projectName* is the name of the speech project in the Speech Project Wizard.

Dialog Designer displays the palette and workspace for the Call Flow Editor.

Overview of the Call Flow Editor

The Call Flow Editor is the primary editor and work place for Dialog Designer. It is where most of the basic work of creating a speech application takes place. The Call Flow Editor contains the workspace where nodes are placed and connected to make up a call flow. It also provides a palette of nodes and options to simple build call flows.

Note:
The Call Flow Editor primary purpose is for creating simple item list call flow logic. For more complex call flow logic, such as looping or calculations for example, Avaya recommends for the developer to write Java code to support these more complex functions.

In the default perspective layout for Dialog Designer, the Call Flow Editor occupies the Editor view space. See Editor View.

Within that view, the Call Flow Editor incorporates the following elements:

- The **workspace area** is the largest area of the Call Flow Editor view. Initially, before building out the speech application, this is the blank area in the upper right area of the view. This area is where nodes are placed and connected.

  During application development, the workspace area can be expanded by clicking the “Maximize” icon in the upper left corner, or by dragging the base of the workspace area. Also, a call flow can be printed at any time using the **File > Print** option.

- The ***applicationName*** **[main.flow]** tab, located at the top of the workspace area is used to bring the main call flow into focus when multiple editors are open.

- The **Application Options Palette**, located along the left side of the view, is used to select nodes to be placed in the workspace. Nodes are grouped as follows:

  – **Template** nodes contain predefined system nodes to be used in an application.

  – **Application items** are used for creating custom nodes to be used in an application.

  In addition, the following functional options are also available within the palette:

  – **Select** is used to select one or more items in the workspace

  – **Connection** is used to connect the nodes and create the flow

  – **Label** is used to add notes and comments to the workspace

  - **Bookmark** is used to create markers so it is easier to locate specific places in larger, more complex applications

  For more information about this palette, see Call Flow Editor Palette Action Options.

  When a node editor is active, the options palette changes to reflect the options available for that node. See Working with Nodes and Node Editors.

- **Node editor tabs**, located along the bottom of the workspace area, show which node editors are open. These tabs are used to quickly switch between node editors and the main flow.

  When clicking the **Application Flow** tab, the workspace displays the main call flow workspace. The second and third tabs of this example display, a menu node (**OptionsMenu**) and a prompt node (**HelloWorld**), respectively. See Working with Nodes and Node Editors.

  Important:
  The Call Flow Editor by default maintains a backup of the flow document, in the event that the main flow is corrupted. Prior to saving the contents of the main flow, the old main flow file is copied to a backup folder using a “round-robin” file numbering system to maintain multiple backups (default is 2; maximum is 10). The backup files are numbered (for example, main.flow.1, mainflow.2, and so on) but once the maximum number of backups is reached, the numbering sequence starts over. Call Flow Editor backup configurations (that is, enabling or disabling the feature, and setting the number of backups maintained) is controlled in the Call Flow Editor Preferences panel. See Call Flow Editor Preferences.

Setting Global Application Properties: AppRoot Node

When a speech project is created, Dialog Designer automatically creates the **AppRoot** node as the initial node of the call flow. This node is the starting point for all call flows and is the one node that cannot be renamed or deleted.

The **AppRoot** node also provides the capability for setting global capabilities for the speech project. That is, use the **AppRoot** node to set event handlers that the system recognizes and responds to, no matter where the caller is in the call flow.

Some examples of the types of parameters and capabilities that can be set in the **AppRoot** node include:

- You might want the system to perform some type of “cleanup” at the end of each call, no matter where, when, or how the call is terminated. You can use a call disconnect event handler in the **AppRoot** node to redirect the application to a form set up to record data about the call. This data might include how the call was terminated, whether by the caller or the system, and where the caller was in the call flow when the call was terminated.

- You can determine how the system functions whenever a caller does not respond as expected, either by not responding at all or by responding with something that does not match what is expected. You can also assign the system a generic prompt to play to callers when an error situation occurs.
- You might want to provide a set of caller commands to which the system can respond regardless of where the caller is in the call flow. For instance, you might want to provide access to general help information from anywhere or allow the caller to cancel transactions.
- You can set a variety of properties, such as timeouts and speech recognition confidence settings, that are then applied system-wide.

The options and parameters that can be set in the **AppRoot** node are available from the options palette and belong to one of two basic types:

- AppRoot Node Application Items - These include the parameters and options *other than* the event handlers that control how an application responds on a global level.
- AppRoot Node Event Handlers - These include the options that determine how the application handles any exceptions that are not defined within a specific node or field.

AppRoot Node Application Items

Application items, in the **AppRoot** node, include the parameters and options, *other than* the event handlers, that control how an application responds on a global level. In many cases, these are items that support the use of the event handlers or perform actions that event handlers are not designed to handle.

The Application Items in the AppRoot node include:

- **Input Parameter** item - Creates a placeholder for input expected from another module. Use this item when you are creating the project to use as a module. When you place this item in the workspace, Dialog Designer automatically creates a variable with the same name.

  See Input Parameter.

- **Capture Expression** item - Used for capturing VXML data via an ECMA script into Dialog Designer variables.

  See Capture Expression.

- **Prompt** item - Tells what prompt to play when the conditions of the event to which it is attached are met. To use it, you must attach it to another item, specifically, any of the AppRoot Node Event Handlers except the **On Disconnect** item. **Prompt** items cannot stand alone in the **AppRoot** node as they can in other nodes.

  Once attached, you must select a prompt to assign to it from the **Name** drop-down list.

See the following sections:

- Working with Prompts
- Prompt

- **Grammar** item - In the **AppRoot** node, the **Grammar** item works only in conjunction with the **Link** item. Use this item to tell the **Link** item which ASR or DTMF grammar to use.

  See the following sections:

  - Working with Grammars
  - Grammar

- **Link** item - Tells the application which node to go to when the system recognizes a specific spoken or DTMF response from the caller. To use this item, you must specify either a DTMF or ASR response (or both) to be recognized. To use the ASR option, you must attach a **Grammar** item to this item. To use the DTMF option, you might specify the required DTMF combination of key presses in this item or attach a DTMF grammar item.

  See Link.

- **Goto** item - Tells the application which node to go to when the conditions of the event to which the **Goto** item is attached are met. To use the **Goto** item, you must attach it to another item, specifically, any of the AppRoot Node Event Handlers except the **On Disconnect** item. You cannot attach a **Goto** item to an event handler that has a **Throw** item already attached.

  See Goto.

- **Throw** item - Tells the application which event to throw when the conditions of the event to which the **Throw** item is attached are met. To use the **Throw** item, you must attach it to another item, specifically, any of the AppRoot Node Event Handlers except the **On Disconnect** item. You cannot attach a **Throw** item to an event handler that has a **Goto** item already attached.

  See Throw.

- **Property** - Use this item to set properties for ASR, DTMF, and transitional audio prompts at a global level. Examples of the types of properties you can set include:

  - Recognition timeouts
  - Confidence and sensitivity levels
  - Terminating key presses

  See Property.

- **External Property** - Use this item to set custom properties for ASR. These custom properties are sometimes required, for example, in conjunction with some ASR engines, such as some of the Nuance ASR software.

  See External Property and the documentation that came with your ASR software.

AppRoot Node Event Handlers

Event handlers, in the AppRoot node, include options that determine how the application handles exceptions that are not defined within a specific node or field.

The AppRoot node can include the following types of Event Handlers. Also see AppRoot Node Application Items for more information.

AppRoot Node Event Handler

Event Handler	Description
Catch (VXML Events)	Determines how an event meeting specific criteria is to be handled when the event is thrown. For it to function, you must assign to the **Catch** event handler at least one **Prompt**, **Goto**, or **Throw** item.
Exit	In the case where there are more serious errors, consider using the **Exit** event handler to exit the application or to return events back to the calling application (from a module).
No Input	Determines what the application does whenever the caller fails to respond. For it to function, you must assign to the **No Input** event handler at least one **Prompt**, **Goto**, or **Throw** item.
No Match	Determines what the application does whenever the system cannot match the response from a caller with any expected response. For it to function, you must assign to the **No Match** event handler at least one **Prompt**, **Goto**, or **Throw** item.
On Disconnect	Determines how the system responds when the caller hangs up, no matter where in the call flow the disconnection occurs.
Return Event	Returns an event to the calling application. Can be used to propagate an event caught in a module to the calling application, or may be used to throw a different event back to the calling application.
Transfer on Event	

Getting Familiar with Nodes to Build Applications

Dialog Designer offers three types of nodes to build projects. All three types of nodes are available from the Call Flow Editor palette. They are grouped into two or three sets, depending on whether module nodes have been imported into the Dialog Designer workbench.

The following bullets introduce the different types of nodes used to build speech applications. These node types are described in greater detail in the subsequent sections:

- Templates (Composite Nodes) - Composite Nodes contain preset functionality and options, which are used as templates. These nodes are easier to use than the Application Items (basic nodes), but they are also intended to be more limited in scope.
- Application Items (Basic Nodes) - Simpler, more generic nodes. These nodes offer greater flexibility than the other two types in many ways. However, to use effectively, they also require a greater knowledge of programming concepts and terminology.
- Module Nodes - Available only if you have imported speech project modules into the Dialog Designer workbench. These modules can be Dialog Designer projects, Nuance Open Speech Dialog Module (OSDM) modules, or any other Web application that can invoked as a VXML <*subdialog*>. Because these are custom-built modules, they can vary widely in simplicity or complexity and flexibility.

Note:
Dialog Designer supports the following OSDM versions: Address OSDM 2.0.3, Core OSDM 2.0.4, and Name OSDM 2.0.1.

Templates (Composite Nodes)

The Templates (Composite) Nodes contain preset functionality and options, which are used as templates. These nodes are easier to use than the Application Items (basic nodes), but they are also intended to be more limited in scope.

Think of a composite node as being a template for a predetermined use. These nodes are actually Form nodes that Dialog Designer populates with one or more form items, in ways that make it easier to create speech application projects. For example, a **Prompt and Collect** node consists of:

- A **Form** node with a nested **Prompt** item
- A speech **Input** item
- A selection of event handlers

On the Call Flow Editor palette, Dialog Designer displays the composite nodes as the **Templates** group. Composite Nodes have a predefined structure in the node editor, and the following properties:

- **Name** - Should reflect the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Can be anything to add as a reminder or description of the purpose or content of the node. Can also be left blank.

  Dialog Designer uses whatever text entered in this property field for the pop-up hint that appears when the cursor hovers over the node icon in the workspace. If left blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

For more information, including complete and detailed descriptions of the composite nodes (**Templates**), see Detailed Node Descriptions.

Note:
A deleted node will have a marker in the Task Tab and the associated Java class will be marked with a warning marker. The task may be deleted if you want to keep the Java code in the project.

Be aware that the Java code will not be regenerated by Dialog Designer and you may encounter Java errors in future upgrades if the Dialog Designer runtime API changes. In such cases, you will have to manually fix the Java errors.

Application Items (Basic Nodes)

The Application Items basic nodes are relatively open-ended nodes with limited built-in functionality. Usually, a basic node performs a single function, though the node can be modified to become more complex.

On the Call Flow Editor palette, Dialog Designer displays the basic nodes as the **Application Items** group.

Some Basic Nodes have default (sub)flows in the node editor, while others do not. With the exception of the Symbolic node, each of these nodes has the following properties:

- **Name** - Should reflect the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Can be anything you want to add as a reminder or description of the purpose or content of the node. Can also be left blank.

  Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.
- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

For more information, including complete and detailed descriptions of the basic nodes (**Application Items**), see Detailed Node Descriptions.

Note:
A deleted node will have a marker in the Task Tab and the associated Java class will be marked with a warning marker. The task may be deleted if you want to keep the Java code in the project.

Be aware that the Java code will not be regenerated by Dialog Designer and you may encounter Java errors in future upgrades if the Dialog Designer runtime API changes. In such cases, you will have to manually fix the Java errors.

Module Nodes

A module node is actually another speech project that is treated as a secondary call flow, or module. The module node invokes the speech project module and transfers the application focus to that module.

Note:
The name of this group, however, can vary depending on the source of the module. Any modules, for instance, that you create in Dialog Designer are grouped together under the label assigned in the **Category** field in the project properties dialog box, **Dialog Designer** properties. See Project Properties. Other modules, such as Nuance OSDM modules or other custom modules are grouped together under **Modules** or whatever other label designated in the appropriate category fields.

To use a module node, you must first create the speech project to be used as the module. You must then deploy it into the Dialog Designer workspace.

For more information and the procedure to do this, see Deploying Projects as Dialog Designer Modules.

Note:
A deleted node will have a marker in the Task Tab and the associated Java class will be marked with a warning marker. The task may be deleted if you want to keep the Java code in the project.

Be aware that the Java code will not be regenerated by Dialog Designer and you may encounter Java errors in future upgrades if the Dialog Designer runtime API changes. In such cases, you will have to manually fix the Java errors.

Tip:
Dialog Designer uses whatever text entered in the **Comments** property field of a module node for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. You can use this as an indication of what the module does or why you are using it, for instance. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

Call Flow Editor Palette Action Options

Dialog Designer uses nodes in the Call Flow Editor as the basic building blocks for speech application projects. Each node represents a piece of code and a functionality. The Call Flow Editor palette provides access to the nodes and action options you use to build call flows.

To build call flows, the following actions are available within the Call Flow Editor workspace:

- Using the Select Option – Move nodes or edit connector lines that are already in the workspace.
- Using the Connection Option – Connect nodes to create the flow.
- Using the Label Option – Add comments to the workspace.
- Using the Bookmark Option – Place a marker in the workspace that you can use to move the focus to that spot in the workspace.

Using the Select Option

Use the Select option to select one or more objects in the workspace. Objects include both nodes and connectors.

To select:

- A single object, click the object.
- A group of nodes, click and drag to enclose the nodes you want to select.

  Note:
  The click-and-drag method selects only nodes. This method does *not* also select connectors. To add connectors to a selected group, use the following Control-click method.

- Multiple objects that are not grouped together in the workspace, or to include connectors, press and hold the Control key down, and then click each object you want to select.

To fine tune the position of the connector dots or of a node:

1. Click the connector or node you want to reposition.
2. On your keyboard, press the period (.) key.
3. Use the arrow keys on your keyboard to nudge the object where you want the object to be.
4. When the object is where you want, again press the period (.) key.

Using the Connection Option

Use this option to connect nodes with a graphical line, thus creating the flow of the call through the nodes. The arrows, both on the nodes and on the connector itself, indicate the direction in which the flow goes. To quick-enable the Connection option, use the **Shift** key.

To connect two nodes:

1. On the Call Flow Editor palette, click **Connection**.
2. Click the outgoing connection point on the first node.

3. Click the incoming connection point on the node to which you want to connect the first node.

Optionally, once you have connected the two nodes, you can use the **Select** option to redraw the connector line. Redrawing connector lines can help you keep your workspace less cluttered looking.

Item Separator

The item separator allows the designer to separate key areas of the application.

Using the Label Option

Use this option to place comments, instructions, or other text messages directly in the workspace, much like a virtual “sticky note.”

Labels provide the capability to:

- Insert explanations of what a certain group of nodes does.
- Direct comments at fellow team members.
- Make written comments in the workspace for whatever need you might have.

Labels do not become part of the generated code.

To use a label:

1. On the Call Flow Editor palette, click **Label**.

   Note:
   The **Label** item might be “hidden” behind the **Bookmark** item. To bring it forward so you can select it, click the down arrow to the right of the **Bookmark** item.

2. Click in the workspace at the location where you want Dialog Designer to display the label.
3. Enter the desired text inside the label.
4. (Optional) Use the side handles to resize the label as desired.

   Note:
   When you first place a label in the workspace and type text into it, the label box expands the width to accommodate the text. You can drag the handles to adjust the height and the width of the label. Once you manually adjust the height and width of the label, the label no longer expands automatically. Text that does not fit into the space, then, is hidden until you manually adjust the size of the label.

Using the Bookmark Option

Use this action to create “bookmarks,” or location markers, in a call flow. These bookmarks are useful primarily in large, complex call flows. In such call flows, you might have several key sections of the call flow that you want to navigate between. With bookmarks, you can do that quickly and easily.

To place a bookmark in the call flow:

1. On the Call Flow Editor palette, click **Bookmark**.

   Note:
   The **Bookmark** item might be “hidden” behind the **Label** item. To bring the **Bookmark** item forward so you can select it, click the down arrow to the right of the **Label** item.

2. Click in the workspace at the location where you want Dialog Designer to display the bookmark.
3. Enter the text to identify the bookmark inside the “flag” part of the bookmark.

Once your bookmarks are in place, use the **Outline** view to navigate between the bookmarks. Bookmarks do not become part of the generated code.

See Outline View.

Working with Nodes and Node Editors

All nodes must be modified, or edited, before they are of any practical use. Each node has an associated node editor, where you can modify existing flows in the node editor or create new ones.

The node editor for each node has its own palette and set of options. Most nodes also include a default structure in the node details editor which you can modify.

- Naming Nodes
- Modifying Nodes/Opening Node Editors
- Adding Items to a Node (in Node Editor)
- Closing a Node Editor

Note:
A deleted node will have a marker in the Task Tab and the associated Java class will be marked with a warning marker. The task may be deleted if you want to keep the Java code in the project.

Be aware that the Java code will not be regenerated by Dialog Designer and you may encounter Java errors in future upgrades if the Dialog Designer runtime API changes. In such cases, you will have to manually fix the Java errors.

Naming Nodes

Each node must have a unique name within the project. Dialog Designer assigns a generic name automatically when a node is placed in the workspace. To assign a name, click the node name line in the node symbol and type the desired name. Alternately, select a node and then change the name in the **Properties** view **Name** field.

CAUTION:
Do not use double-byte or extended ASCII characters when naming nodes.

Note:
Node names must follow Java naming conventions (see Naming Java Components).

Tip:
Avaya recommends that you assign a name that communicates the central purpose or function of the node.

When a node is named and a project is saved, Dialog Designer automatically names the Java class associated with that node the same name as the node. If later the node is renamed, Dialog Designer again changes the name of the associated Java class, as long as the call flow is also saved at the same time.

Note:
Be sure to also save the call flow when changing a node name, or the Java class can get out of sync with the rest of the application.

Modifying Nodes/Opening Node Editors

To modify a node, the node editor for that node must be opened by double-clicking on the node. When the node editor is open, the (sub)flow contained for that node can be created or edited.

All node editors open in the same workspace as the Call Flow Editor. When the node editor opens, Dialog Designer displays its tab at the bottom of the Call Flow Editor view. This tab provides a way to switch quickly between node editors and the main flow in the workspace.

Each node editor includes a palette of options and items for use with that node, similar to the Application Options palette in the main flow workspace.

See Nodes and Palette Options.

Note:
A deleted node will have a marker in the Task Tab and the associated Java class will be marked with a warning marker. The task may be deleted if you want to keep the Java code in the project.

Be aware that the Java code will not be regenerated by Dialog Designer and you may encounter Java errors in future upgrades if the Dialog Designer runtime API changes. In such cases, you will have to manually fix the Java errors.

Adding Items to a Node (in Node Editor)

To add an item to the node editor workspace, first click the item in the palette and then click-and-drag the cursor in the workspace to the desired location. This procedure is similar to the method of adding nodes to the main call flow, but there are a couple key differences in the node editors:

- In node editors, click and drag to place the items, in most cases.
- In node editors, items cannot be placed wherever you want them to go. Sometimes, they cannot be placed at all.

  To know whether an item can be placed in the node, note the appearance of the cursor. When an item can be placed in the node editor, the cursor changes to display a plus sign (+).
- Many node items depend on other items. For example, a **Grammar** item cannot be used by itself. A **Grammar** item must always be attached to another item, as a sub-item to that item.

  For more information about the dependencies and restrictions on items, see Nodes and Palette Options.
- Items do not need to be connected in the node editor. Usually, items are executed in the order in which they appear, though there are exceptions. For example, event handlers only execute when the conditions for their execution are met.

Closing a Node Editor

To close a node editor, click the small **x** box in the upper-right corner of the workspace. It is located just underneath the sizing button for the Call Flow Editor.

Working with Sub Flows (vs. Modules)

When designing your speech application, design considerations need to be taken into account for when and where it is better to create sub flows or reusable modules in your application. To best understand the differences, refer to the following table for side-by-side considerations.

Modules	Sub Flows

• Project resources are self-contained.
• Reusable across projects.
• Better for multiple developers.
  ◦ Developers can work separately on their modules with out much concern that it will impact other developer's work.
• Separate application contexts at runtime.
  ◦ Session data is independent.
  (DD runtime has mechanisms to share Java objects across modules).
  ◦ Voice browser fetches as VXML <*subdialog*>.
	• Share project resources (prompts, grammars, variables, etc.).
	• Reusable only within a project.
	• Not good for multiple developers.
	  ◦ Multiple project files are shared and updated by various editors (variables, web.xml, etc.). Trying to merge updates to these files is difficult.
	• Same application context at runtime.
	  ◦ Share session data.
	  ◦ Requests forwarded entering/exiting sub flows.
	  ◦ All VXML is part of the same dialog.

In general, creating and working with Sub Flows is very similar to creating and working with main flows. However, following are a few differences:

- Palettes are similar with all modules listed. Return items are different however.
- Sub flows have a Begin node which is the entry point to the flow. There is no “app root” on a Sub Flow. AppRoot is defined in main.flow.
- Sub Flows support multiple return points, forwarding the request to the next node in the calling call flow. Main flows only allow one return (exit).

Some considerations about sub flows:

- Always contained within the same application scope as the main flow.
- Share project variables
- Share resources, such as prompts, grammars, DB operations, WS operations, etc.
- Share application context (and session objects at runtime).
- Sub flows can be nested (i.e., call other sub flows or call modules).
- There is no limit to the number of sub flows that can be in a project; however they are not intended to replace modules. The developer should design the application appropriately with proper common sense when designing large applications.

- Implicit variable items (e.g., Menu, Prompt & Collect, Transfer items, etc.) names must be unique across all flows in a project. When adding/renaming these items, the Call Flow Editor and project variables will enforce uniqueness.
- Sub flow code is generated in a separate package. For example, “MySubflow.flow” will generate Java code in “flow.subflow.MySubflow” package.
- Servlet names/mappings changed (web.xml). For example, “MySubflow.flow” will have servlet names starting with “MySubflow-<*node name*>"; servlet mappings start with “/MySubflow-<*node name*>”.
- With sub flows, there should be no impact on runtime performance. Servlets are all part of the same web application, and requests are forwarded when entering/returning from sub flow.

For more details on working with Sub Flows, see the following sections:

- Creating a Sub Flow Using the Sub Flow Wizard
- Invoking a Sub Flow
- Copying, Cutting, and/or Pasting Sub Flows
- Refactoring Projects to Use Sub Flows

Creating a Sub Flow Using the Sub Flow Wizard

To create a sub flow using the sub flow Wizard:

1. Create a new call flow file using one of the following methods:
   - Go to: **File > New > Call Flow file**
   - Use the Toolbar Button
2. Enter a unique name of the Sub flow name.
3. A new **.flow** file is created in the project under the **flow** directory.

Invoking a Sub Flow

To add a subflow node:

1. In the Properties view, select the call flow to call.
   - A combo box will list all of the sub flows in the project.
   - Cannot select main.flow; sub flows cannot select themselves.
2. When a sub flow is selected, the return points are added to the node.
   - Each return point is used to draw the flow to the next node when returning from the sub flow.

- Return points are added/renamed/removed when changed in the sub flow.

3. Multiple return points allow you to return back from a sub flow and execute different parts of the calling flow depending on the return point that was used.
   - Something like a “switch” statement in programming.

Copying, Cutting, and/or Pasting Sub Flows

When copying, cutting, and/or pasting Sub Flows, refer to these considerations:

- Copying, cutting, and/or pasting is supported in main and Sub Flows.
- Copying or pasting from one flow to another flow in the same project will move nodes.
  - Does not delete or rename variables.
  - Cannot cut and paste within the same flow.
- Cutting or pasting from one flow to another flow in the same project will duplicate nodes.
  - Will create copies of nodes & variables.
  - If a variable is renamed (to avoid name conflicts) when pasted, then all references to the variable in the copied selection will be updated to reference the name of the new variable.

  Important:
  Copying other nodes that reference a variable that was renamed during an earlier copy/paste operation may not reference the right variable when pasted. For example, if a Prompt & Collect (“AccountNum”) is copied and renamed (“AccountNumCopy”), then all nodes in the selection will change to reference “AccountNumCopy”.

  If you later copy other node(s) that reference “AccountNum”, then they will continue to reference “AccountNum” when you paste. This is because at this point the project would have both “AccountNum” and “AccountNumCopy” variables.
- Copying a Sub Flow reference node to another project will copy the Sub Flow and all nested resources to the target project.

Refactoring Projects to Use Sub Flows

When refactoring projects to use Sub Flows, review the following considerations before performing your refactoring efforts:

- Move nodes (cut and paste) between main flow and sub flows. This preserves variable names and references and will not create duplicates (clones).
- Some nodes cannot be moved to a Sub Flow. AppRoot and Return (main.flow) are not allowed on Sub Flows. Do not include them in the selection when moving to the Sub Flow.
- Symbolic node references may be cleared. (Cleared if symbolic node is copied, but the reference node is not.)
- Plan the return points from the Sub Flow, as follows:
  - A Sub Flow may utilize multiple return points in order to handle different return paths in the calling flow.
  - When selecting a group of nodes to move to a Sub Flow, often a symbolic node indicates a need for a return point when the reference node is not in the selection.

  Important:
  Keep a backup so you can revert if you encounter problems.
- Renaming a Sub Flow is supported, but not encouraged. **Never** rename “main.flow”.
- Copying/Cutting from main flow to a Sub Flow will copy/move Java code. User-defined code may have compile errors if referencing code in “flow” package without fully qualified name (e.g., Project Variables).
- Imports must be organized to fix import statements:
  - Open Java class: Source > Organize Imports (Ctrl+Shift+O).
  - From Package/Project Explorer view, right click on the Java class: Source > Organize Imports
- Cut and paste in same flow is no longer supported
  - Cut and Paste in same flow is just a move operation.
  - Was not recommended before because it would clear variable references.

  Important:
  Be careful for infinite loops. Do not create circular references to sub flows.

Using the Reusable Module Import Wizard

Dialog Designer reusable modules can be reused in more than one speech application. This can be very useful for common project elements needed in more than one application. It can also be useful when using a speech module that uses the same phrases as the parent application.

To import Dialog Designer reusable modules to be used in another speech application projects, the reusable module(s) must first be exported from a different project. See Using the Export Dialog Designer Project Wizard.

To import a Dialog Designer reusable module, use the Import Dialog Designer Reusable Module Wizard.

Access the wizard using one of the following methodologies:

- From the **File** menu, select **Import...**, then select **Avaya Speech Development > Import Dialog Designer Reusable Module**.
- Right-click anywhere in the **Navigator** view, then select **Import...**, then select **Avaya Speech Development > Import Dialog Designer Reusable Module**.

Select the appropriate options within the Import Wizard.

Import Dialog Designer Reusable Module Wizard—Specify Import Details

Option	Description
Import VoiceXML subdialog from Web Archive (WAR) file	
Import VoiceXML subdialog from Web Archive (WAR) file	Select this option to import VoiceXML subdialog for a Web Archive (WAR) file. After selecting, define the WAR file to import, type, and number of reusable dialogs contained in the archive, then **Next**, the VoiceXML is imported.
Select WAR file	Click the **Browse** button to navigate to the WAR file to be imported.
Type	Indicate the type of WAR file to import VoiceXML from. Available types are: • Dialog Designer reusable dialog (module) • Nuance OSM - If selected, • Other/unknown (not supported)
Number of reusable dialogs contained in archive	Indicate the number of reusable dialogs contained in the archive to be imported.

Import Dialog Designer Reusable Module Wizard—Specify Import Details (continued)

Option	Description
Manually define module definition	
Manually define module definition	Select this option to create a module definition for a VoiceXML subdialog without importing a WAR file. This can be used to integrate third party VoiceXML subdialogs, such as Nuance OSDMs. External URLs are supported, but they cannot be modified after the module definition has been created. After selecting, click **Next** to define the module details, as follows: • **Name** - Name of the subdialog to be created. For OSDMs on external servers, the name must start with “OSDM_”. Record must be “OSDM_Record”. • **Version** - Version numbering of the subdialog to be created. • **URL** - URL of the Servlet/JSP/VXML sub-dialog for the module being created. This URL is mandatory. **Note**: Once the URL is created, it cannot be changed through the Import Wizard; only delete and add a new one, if needed. • **Category** - Indicate a category of the subdialog, if applicable. When used, the module is added to the Call Flow palette (for grouping). • **Vendor** - Indicate the third-party integration vendor. For OSDMs, the vendor must be “Nuance”. • **Description** - Enter a description to describe the subdialog to be created. • **Input Parameters** - Click **Add** to add and define input parameters, values, and descriptions. For OSDM support, add parameters are specified in the OSDM documentation. • **Output Parameters** - Click **Add** to add and define input parameters, values, and descriptions. For OSDM support, add parameters are specified in the OSDM documentation.

Chapter 7: Working with Phrases and Phrasesets

One of the key elements used to create speech applications with Dialog Designer is the *phrase*. Phrases are used to build prompts, which in turn are used to guide the caller through the call flow. Phrasesets are used to group phrases, usually all related to a particular speech application.

This chapter describes all aspects of creating and using phrasesets in Dialog Designer. It contains the following sections:

Important:
Phrases or phrasesets are only used with speech applications. Call control applications do not incorporate phrases or phrasesets.

About Phrases

In the world of speech and interactive voice response (IVR) applications, phrases are among the most basic building blocks. They consist of prerecorded audio files.

They can be as short as one word or as long as several complete sentences. Or they might even be music recordings, as is often the case with transitional audio prompts. See About Transitional Audio Prompts.

Although Text-to-Speech technology can be used to render audible output to callers, prerecorded audio phrases are usually preferable whenever possible. Audio phrases give a finished speech application a more professional presentation and a more satisfying experience for the caller.

See the following sections:

How Phrases Are Used

Usually, phrases are used as prompt segments and are assembled together to build prompts. For example, at a certain point in a call, the system might use a prompt to report to the caller the balance in a checking account belonging to the caller. The final prompt might say something like: “You have $528.48 in your checking account.”

This one prompt, however, might consist of as many as eight or more separate phrases:

- “You have”
- “five hundred”
- “twenty-eight”
- “dollars”
- “and”
- “forty-eight”
- “cents”
- “in your checking account.”

Because a return statement from the system often must use variable values, as with the money amount in the example, the number and currency words can be contained within and used as variables.

Where Phrases Are Used

Phrases are accessible only in the Prompt File Editor. The Prompt File Editor is where you assemble phrase files and other prompt segments into the prompts for use in the speech application.

See Using the Prompt File Editor.

Types of Phrases

Dialog Designer uses two types of phrases: standard phrases and custom phrases. See the following sections for more information on these types of phrases:

- About Custom Phrases

About Standard Phrases

Standard phrases are included with Dialog Designer as part of the language phrase bundles purchased. They are not, however, installed automatically. If you want to use them, you must first install them.

Note:
Although there is nothing to prevent this, standard phrases are not intended for general use in an application. Their primary intended use is to be associated with the localization bundle and audio variables. If they are used and the localization bundle is later altered, unexpected results may be encountered.

Standard phrases consist of common types of phrases used in prompts. They include numbers in various formats, currency words, names of months and days of the week, years, and similar words and phrases.

Standard phrases are easily identified in the Navigator view because the names of standard phrase audio files start with the characters **std**_.

Note:
Since Dialog Designer uses the **std**_ prefix to distinguish standard phrases, do not use the **std**_ prefix when naming custom phrase files.

For information about installing standard phrases, see Using a Localization Bundle's Standard Phrasesets.

Note:
Standard phrases were converted into a phraseset with Dialog Designer 3.1.

About Custom Phrases

Custom phrases are phrase files that can be created with audio files recorded by you, or by a professional voice talent. They can be long or short and can also include periods of silence.

While Dialog Designer provides the capability to record custom phrases within the interface, Avaya recommends engaging the services of a professional voice talent and recording studio to record the phrase files for a finished application. With professional studio recordings, phrases audio recordings will be of higher quality, and can be better tuned, to sound better and to trim silence from the beginning or end of recorded files. Also, a professional talent can better match the standard phrase voice, which enables a more seamless prompt to callers.

Note:
Professionally recorded audio files must be transcoded to one of the formats required by the VoiceXML 2.0 standard. See the "Audio File Formats" appendix in the W3C VoiceXML 2.0 Recommendation.

About Phrasesets

Phrasesets are used to group phrases, usually all related to a particular speech application. Phrases are used to build prompts, which in turn are used to guide the caller through the call flow.

The main advantage for grouping phrases, aside from better organization is that with phrasesets, file resource efficiencies are greatly improved which translates into quicker build times. With a phrase, there are four files per phrase (one .phrase file, one .java file, one java .class file, and one .wav file for every phrase), so when there are 1000 phrases in a project, there are actually 4000 files that need to be managed and built. With a phraseset, there can be 1000 phrases in a phraseset, and so in this example, there are only 1003 files to manage (.phraseset file, one .java file, one Java .class file, and 1000 audio files).

Phrasesets also support referencing audio files stored on an external server.

Using the Phraseset File Wizard to Create Phraseset

Phrasesets are used to group phrases, usually all related to a particular speech application. To create a phrase, see Using the Phraseset Editor.

To create a phraseset, use the Phraseset File Wizard. See the following sections:

- Accessing the Phraseset File Wizard
- Create a Phraseset Page
- Specify Audio File Directory Page

After a phraseset file is created, to record an audio file or to set other properties for the phraseset file, use the Phraseset File Editor. See Using the Phraseset Editor.

Accessing the Phraseset File Wizard

To access the Phraseset File Wizard, use one of the following methodologies:

- From the **File** menu select **New** > **Phraseset File**.
- On the main toolbar, click the icon for the phraseset file wizard ().
- Right-click anywhere in the Navigator view, then select **New** > **Phraseset File**.

Dialog Designer displays the Create a Phraseset Page page to begin defining the phraseset.

Create a Phraseset Page

To create a new phraseset, configure the following settings in the Create a Phraseset page of the Phraseset File Wizard.

Phraseset File Wizard—Create a Phraseset Page Settings

Setting	Description	Required?
Available Projects	Select the project to which the phraseset file will be assigned.	Yes
Available Languages	Select the language(s) in which this phraseset will be available. This pane is populated with all the languages existing in your workspace. See Administering Project Languages.	Yes
Phraseset name	Name identifying the phraseset that intuitively describes the content of the phraseset file (or project to use the phraseset, if more applicable). Dialog Designer automatically adds a **.phraseset** extension to the file name when the phraseset file is created. **Note:** Phraseset file names must follow Java naming conventions (see Naming Java Components). **CAUTION:** Do not use double-byte or extended ASCII characters when naming the phraseset.	No, but recommended. If a name is not entered, a generic name is assigned by Dialog Designer.

When done, perform one of the following actions:

- To specify the directory where the audio files that will be added to the phraseset file reside, click **Next**. Dialog Designer displays the Specify Audio File Directory Page.
- To exit the Phraseset File Wizard without creating the phraseset file, click **Cancel**.

Specify Audio File Directory Page

In the Specify Audio File Directory page of the Phraseset File Wizard, enter the directory path (or click **Browse** to navigate) to the appropriate path where the audio files (phrase files) that will be added to the phraseset file reside.

When done, click **Finish**.

Dialog Designer creates the phraseset file and automatically opens a Phraseset File Editor used to add (or delete) additional phrases to the phraseset file, set other properties for the phrase files within the phraseset file, or record an associated audio file. See Using the Phraseset Editor.

Using the Phraseset Editor

Use the Phraseset File Editor to modify phraseset files once they are created. When opened, the Phraseset Editor is displayed in the main editor workspace of Dialog Designer. It has three page tabs associated with it— Phrases, Phrase, and Audio Tabs—that are used for the following tasks, respectively:

- Creating, adding, or deleting phrases to an already created phraseset file.
- Setting or changing text descriptions, comments, search keywords, and other attributes for the phrase files within the phraseset file.
- Setting or changing parameters for an associated audio file, or to record a new audio file.

See the following sections for more information on using the Phraseset Editor:

- Accessing the Phraseset Editor
- Phraseset Editor—Add New Phrase Dialog
- Phraseset Editor—Phrase Data Tab
- Phraseset Editor—Audio Tab

Accessing the Phraseset Editor

To access the Phraseset Editor, use one of the following methodologies:

- Create a phraseset file with the Phraseset File Wizard, which subsequently opens the Phraseset Editor after clicking **Finish** to create the phraseset file. See Using the Phraseset File Wizard to Create Phraseset.
- In the Navigator view, double-click the ***.phraseset** file to edit, or right-click the ***.phraseset** file and select **Open With > Phraseset Editor**.
- For a phraseset file that is already open in the Phraseset Editor workspace but not currently showing in the active workspace, click the Phraseset Editor tab for that phraseset file.

Phraseset Editor—Add New Phrase Dialog

There are also sections for specifying if the audio files are stored locally or externally.

Upon displaying the Phraseset Editor, all phrases that are currently included within the phraseset are shown in the left panel of the editor. Within this tab, the following options are available:

- To set Phraseset Audio Files Location, select **Local** or **External** (and enter a URL base) for where the phraseset audio files will reside.
- To delete a phrase within the phraseset, select it, then click **Delete Phrase**.
- To add additional phrases to the phraseset, click **Add New**. A dialog is displayed to name the Phrase, and basic audio information properties. The following table describes the settings. Additional settings may be configured by selecting the newly created phrase, and editing within the Phraseset Editor—Phrase Data Tab.

Phraseset Editor—Add New Phrase Dialog Settings

Setting	Description	Required?
Phrase name	Enter a phrase name for the new phrase to be added to the phrase set.	No
Audio Information Properties		
Audio File: • **New** • **Existing**	Select one option: • To create a new audio file, select **New**. An audio file format, or **Extension**, must also be selected. • To use an existing audio file, select **Existing**. Dialog Designer enables the **[File name] Browse** button to navigate to the file to use. All other **Audio Information** properties are disabled.	Yes, if available.
Extension: • **wav** • **au**	If **New** is selected as the **Audio File** type, select the appropriate audio format for the audio file. The default **wav** file format is usually used on Windows-based computers, and the **au** format is more commonly used on Sun and UNIX computers. **Note:** Professionally recorded audio files must be transcoded to one of the formats required by the VoiceXML 2.0 standard. See the “Audio File Formats” appendix in the W3C VoiceXML 2.0 Recommendation.	Yes, if **New** is selected.

Phraseset Editor—Phrase Data Tab

Use the Phrase Tab of the Phraseset Editor to set basic identification properties for the phraseset file. This tab is located at the bottom of the workspace area when the Phraseset Editor is open.

In the Phraseset tab, the following attributes of the phraseset file can be configured.

Phraseset Editor—Phrase Tab Attributes

Attribute	Description	Required?
Display Name	A name for the phraseset file.	No
Voice Talent	Information about the professional individual or studio that recorded the phraseset audio file, if applicable.	No
Phrase Text	Exact text for the phraseset file. This field serves three purposes, as follows: • It identifies the intended wording and purpose for the phraseset file. • If no audio file is selected or exists for the phraseset file, the system automatically uses this text to render the phraseset using text-to-speech (TTS) technology. • If the audio file is planned to be recorded later, this text serves as a script for the individual or company who will do the recording.	No, but strongly recommended
Base Language Text	Exact text for the phraseset in the default project language. Often, this contains the same text as shown in the **Phraseset Text** field. However, when creating applications that will be converted or translated into another language, these fields will contain different information. For example, suppose the application is developed in English, but will be translated to German. In such a case, the **Base Language Text** contains the text in the language that the application is developed in. For example, if English, the text could be: “Good morning. Thank you for calling the German Federal Bank.” The **Phraseset Text** field, however, contains the German translation of the English **Base Language Text** phraseset: “Guten morgen. Danke fuer Ihre Anruf zum Bundesbank Deutschland.”	No

Phraseset Editor—Phrase Tab Attributes (continued)

Attribute	Description	Required?
Comments	Descriptive comments related to the phraseset file. This text is only displayed in the Phraseset Editor.	No
Search Keywords	Key words that can be used when searching among phrasesets. **Note:** The current release of Dialog Designer does not support the ability to search based on keywords. This capability will be supported in a future release.	No

Phraseset Editor—Audio Tab

Use the **Audio** tab of the Phraseset Editor to select an audio file or record a new audio file for the phraseset file. An option to listen to the currently assigned audio file is also available. This tab is located at the bottom of the workspace area when the Phraseset Editor is open.

Note:
To record an audio file, a microphone must plugged in to your computer. Before recording, properly calibrate the automatic record levels for the microphone. Also, it is recommended to use headphones while recording, rather than rely on speakers connected to the computer. See Configuring the Microsoft Speech SDK for Microphone Input.

The **Audio** tab has the following attributes and options.

Phraseset Editor—Audio Tab Attributes and Options

Attribute/Button	Description
File Name	(Display only) Displays the name of the file that contains the audio recording assigned to the phraseset file.
Browse (local audio only)	To change the currently assigned audio file, use this button to locate a different prerecorded audio file. The audio file must be either *.wav or *.au format (local audio only).
Test (external audio only)	Attempts to fetch the audio file from the external server.
Format	(Display only) Displays the audio format of the selected audio file. **Note:** The current release of Dialog Designer supports only 8KHz 8-bit mono formats.
Position	(Display only) Indicates the position of the playback cursor while an audio file is being played back.

Phraseset Editor—Audio Tab Attributes and Options (continued)

Attribute/Button	Description
Length	(Display only) Indicates the length of the audio file, in seconds.
Play button 	Click this button to play the audio file. **Note:** This button is inactive until you record or import an audio file, or fetch an external audio file with the **Test** button.
Stop button 	Click to end a recording or stop an audio file that is being played back. **Note:** This button is active only when recording or playing back a recording.
Record button 	Click to start recording an audio file. **Note:** Recording is not supported for external audio phrases.

Using the Recording Script Wizard

Dialog Designer localization features allow Dialog Designer developers to switch between different languages to play an application. Often, Dialog Designer developers need to produce a report for a recording studio to record all phrases of an application in a new language.

Using the Recording Script Wizard, developers can quickly produce a cleanly formatted HTML report of all phrases in a speech application for submitting to recording studio. The report contains the following details:

- project name
- language
- audio file format
- if applicable, any specified comments or notes to the recording studio about the phrases
- date
- a list of phrases (all phrases, or if appropriate, categorized as user defined phrases and standard phrases) including:
  - each phrase's audio file name
  - text to be recorded
  - pronunciation of the phrase

See the following sections for more information on using the Phrase Editor:

- Accessing the Recording Script Wizard
- Creating a Recording Script Page
- Specifying Script Parameters Page

Accessing the Recording Script Wizard

To access the Recording Script Wizard:

1. In the Navigator view, select a project for which to create a recording script report.
2. From the menu bar, select **Project > Reports > Recording Script**.

Dialog Designer displays the Creating a Recording Script Page of the Recording Script Wizard to begin defining the phrase.

Creating a Recording Script Page

Complete the following settings in the Specify Audio Parameters page of the Phrase File Wizard.

Recording Script Wizard—Create a Recording Script Page Settings

Setting	Description
Available Project	Select the project for which a recording script report will be created.
Language	Select the language for which a recording script report will be created.
Phrase Type	Select the types of phrases that the recording script report should contain. Options are: • All Phrases • Used Defined Phrases (also called Custom Phrases) • Standard Phrases
Select the report destination directory	Enter a path where the recording script report will be saved. Alternatively, click **Browse** to navigate to an appropriate directory. The file name of the saved report uses the following format: ***ProjectName_SelectedLanguage*_recordingscript.html**

When done, perform one of the following actions:

- To create the recording script report without further definition of audio file formats or any additional comments to be added within the report, click **Finish**.

  Dialog Designer creates and saves the recording script report file containing, by default, **Raw (headerless) 8kHz 8-bit mono mu-law** audio files or the last chosen audio file format, if prior script reports have been run.

  Double-click on the saved HTML report file where saved to view it.

- To define the audio file format and additional comments to be added within the report, click **Back**. Dialog Designer displays the Specifying Script Parameters Page.
- To exit the Recording Script Wizard without creating the recording script report file, click **Cancel**.

Specifying Script Parameters Page

Complete the following settings in the Specify Script Parameters page of the Recording Script Wizard.

Recording Script Wizard—Specify Script Parameters Page Settings

Setting	Description
Audio File Format	Select the audio file format of the phrases that should be included in the recording script report. Options are: • Raw (headerless) 8kHz 8-bit mono mu-law • Raw (headerless) 8kHz 8 bit mono A-law • WAV (RIFF header) 8kHz 8-bit mono mu-law • WAV (RIFF header) 8kHz 8-bit mono A-law By default, **Raw (headerless) 8kHz 8-bit mono mu-law** audio files are chosen, or the last chosen audio file format, if prior script reports have been run.
Recording Comments	Free text field to add comments or instructions to the recording studio who will ultimately receive the recording script report.

When done, perform one of the following actions:

- To create the recording script report, click **Finish**.

  Dialog Designer creates and saves the recording script report file containing, by default, **Raw (headerless) 8kHz 8-bit mono mu-law** audio files or the last chosen audio file format, if prior script reports have been run.

  Double-click on the saved HTML report file where saved to view it.

- To exit the Recording Script Wizard without creating the recording script report file, click **Cancel**.

Exporting and Importing Phrase Files

In Dialog Designer, phrases from one speech application can be reused in another speech application projects. This is useful when a common set of phrases are used in two or more applications, or when you have a speech module that uses the same phrases as the parent application.

See the following sections for more information:

- Exporting a Phrases Zip File
- Importing a Phrases Zip Files
- Importing a Delimited Phrase Data Text File

Exporting a Phrases Zip File

To export a Phrases Zip File so they can be reused in more than one speech application project, use the Export Phrases Wizard. To access the wizard, use one of the following methodologies:

- From the **File** menu, select **Export...**
- Right-click anywhere in the **Navigator** view, then select **Export...**

Important:
Be careful when exporting phrase files. Prompts typically rely on phrases, grammars, and other project resources that may not exist in the project you are exporting the phrases to. If you import phrase files that rely on other resources into other projects, Dialog Designer generates errors. Either export the missing resources for the other project or redirect the imported phrases to make use of other resources in that other project.

Select the following appropriate options within the Phrases Export Wizard.

Phrases Export Wizard

Option	Description
Select Page	
Select an export destination	Select **Avaya Speech Development > Phrases Zip File**, then click **Next**.

Phrases Export Wizard (continued)

Option	Description
Specify Export Parameters Page	
Available Projects	Select the project from which to export phrases.
Available Languages	Select the language(s) for which you want to export the phrases. This field lists only those languages for which phrases exist in the selected project.
Zip file	Perform one of the following actions: ● Enter the path and filename to be assigned to the exported .zip file. ● Use **Browse** to specify where the .zip file will be exported.
Options	Optionally: ● Select the **Include directory structure** checkbox to include directory path information within the .zip file. ● Select **Include project name** checkbox to include the project name from where the phrases are being exported.

Importing a Phrases Zip Files

To import a Phrases Zip File into a speech application project, a file must first be exported from another project. The exported file must then be imported into the project where they will be used.

Important:
Be careful when importing prompt files. Prompts typically rely on phrases, grammars, and other project resources that may not exist in the project where they are being imported. If you import prompt files that rely on other resources, Dialog Designer generates errors to let you know what resources are missing. You must then either import the missing resources or redirect the imported prompts to make use of other resources.

Note:
The following procedure applies only to prompt files that were exported to a Zip file using the Export wizard in Dialog Designer.

To access the Import Phrases wizard, use one of the following methodologies:

- From the **File** menu, select **Import...**
- Right-click anywhere in the **Navigator** view, then select **Import...**

Select the following appropriate options within the Phrases Import Wizard.

Phrase File Import Wizard

Option	Description
Select Page	
Select an import destination	Select **Avaya Speech Development > Prompts Zip File**, then click **Next**.
Phrases Page	
Phrase zip file	Perform one of the following actions: • Enter the full path to and name of the .zip file to be imported. • Use **Browse** to locate and select the .zip file to be imported.
Available Projects	Select the project for which the prompts will be imported.
Overwrite existing resources without warning	To overwrite any existing prompts files with the same names in the project, without warning, select this check box.

Importing a Delimited Phrase Data Text File

The Import Phrase Data option allows you to import phrase data from a delimited text file to update existing phrase data, and/or add new phrase data, to the phraseset within your speech application project.

Before importing a phrase data file, you need to first create one—a delimited phrase data text file. The column layout of the fields in the text file is configurable during the import. The rows, however, must have a consistent number of columns, whether each column in a row is defined data or left blank.

The delimiter used in the text file can be a Tab, or another consistently used character in your file that does not conflict with the data content; for example, a pipe symbol (“ | “) or comma.

The delimited phrase data text file can contain some or all of the following accepted data content within its columns, on a row by row basis:

- Audio file name
- Phrase name
- Phrase text
- Comments
- Voice talent
- Base language text

- Search keywords

Following is an example of a short delimited phrase file:

```
MainMenu.wav|Main Menu|This phrase describes the main menu|James Earl Jones
Directions.wav||This phrase includes directions to the store|
StoreHours.wav|Hours||Valerie Snow
```

With this example delimited phrase file, the columns are configured as follows within the Import Phrase Data page. The remaining data content types are not used:

```
audio file name|phrase name|phrase text|voice talent
```

New phrases included in the text file can also be added, or ignored, as specified in the **Import Phrase Data** page. Also, a data index area on the page allows for a controlled and configurable import with options to specify what text row from the file to begin importing data from, as well as a place to define a zero-based index of the columns.

To import a delimited phrase data file:

1. Create a delimited phrase data file, as described in the previous discussion.
2. Within the *Dialog Designer Navigator* panel, select the speech application project to import the phrase data to.
3. From the *Main Menu* bar, select **File > Import**. The *Import selection* dialog is shown.
4. Select **Avaya Speech Development > Phrase Data**, then click **Next**.
5. The *Import Phrase Data* dialog is displayed. Click **Browse** to navigate to your delimited phrase data file to be imported.
6. In the *Target Phraseset* panel, select a **Project**, **Language**, and **Phraseset** which will be updated based on the data within the phrase data file.
7. In the *Options* panel, determine the following options:
   - **New phrases** – Select either **Ignore** or **Add** to indicate whether new phrase data should be ignored or added to the phraseset, respectively.
   - **Delimiter** – Depending on the delimiter used in your phrase data file, select either the **Tab** option, or **Other** and indicate the delimiter character.
8. Click the **Advanced Options** label, and options to configure the column layout of the delimited phrase data file are provided (if required). Using these options, configure the import based on the defined phrase data file, as follows:
   - If appropriate, define the **Start import at row** option. This option allows you to define either 0, for the import to start at the beginning of the phrase data file, or another number, to start at a row later within the file.

     This feature is especially useful if you use the same delimited phrase file again and again over time and you only want to bring in new data, not overwrite existing data.

- Define values to the available data content fields, based on the data defined in the columns of the delimited phrase data file. The index to be defined is a zero-based (zero being the first column) index, and counts up for the column order for data included in the text file. If a particular type of data is not included or should not be imported, use a value of -1.

9. When done, click **Finish**.

Chapter 8: Working with Prompts

One of the key elements used to create speech applications with Dialog Designer is the *prompt*. Prompts are used to guide the caller through a call flow. Prompts play announcements to the caller, prompt the caller for some response, or respond to the request of a caller for information.

This chapter describes all aspects of creating and using prompts in Dialog Designer. It contains the following sections:

> **Important:**
> Prompts are only used with speech applications. Call control applications do not incorporate prompts.

About Prompts in Dialog Designer

In the world of speech and interactive voice response (IVR) applications, prompts are among the most important building blocks. A prompt consists of a set of one or more *prompt segments* that are assembled to communicate information to a caller and encourage the caller to respond to a request for input.

See the following sections:

Prompt Segment Types

Prompts consist of any combination of following types of prompt segments:

- **Phrases** – Prerecorded digital audio files that the system plays back. See About Phrases and Phrase Variable.
- **Audio variables** – Special variables that analyze the content of the variable, parse the content, and play the content back using special prerecorded phrase files. See Audio Variable.
- **Phrase variable** - Special variable for playing project phrases where the variable value is a phrase name or audio file URL. These types also support failover TTS that is hard coded or stored in a variable. Also supports collections where the variable is a collection of phrase names.
- **Text variables** – Variables that contain text which the system can render as synthesized speech using TTS technology. See Text Variable.
- **Text-to-Speech (TTS)** – Text entered in the **TTS Text** field of the TTS prompt item and which then renders the content as synthesized speech using TTS technology. See TTS.
- **Expression segments** – ECMAScript expression segments that are typed in and rendered into a generated VXML page. Expression segments can be useful for playing certain VXML page variables in a prompt.

  To properly use an expression in a prompt, consult the W3C VoiceXML 2.0 Recommendation so that valid expressions are used properly (since different expressions are legal—or illegal—at different points in a VXML page).

Prompt Controls

In addition to selecting prompt segments from among the available types, Dialog Designer provides a number of different prompt controls that can be used to control different aspects of a prompt in use. For example:

- Use prompt levels to vary the prompt that is played to the caller, based on how many times the prompt is repeated. See Understanding Prompt Levels.
- Determine the order that prompt levels should play back. See About Play Order for Prompt Levels.
- Use condition settings to control which prompt segments play and under what circumstances. See About Conditions in Prompts.
- Control the rendering of TTS with the use of SSML tags. See About SSML Controls in Prompts.
- Determine whether and how barge-in can be used with the prompt. See Prompt File Editor—Prompt Main Tab.

Understanding Prompt Levels

When a caller does not respond to a prompt at all or the ASR server does not recognize the response, the system usually repeats the prompt. In some speech applications, there might be reason to repeat the prompt content, but with a variation to the first prompt. The system can be configured to prompt the caller the second time using a slightly different version of the prompt.

For example, suppose a speech application has a prompt that says:

> "Please state your seven-digit account number. The account number is located in the upper-right corner of your bill."

If the caller says the number too quickly for the ASR server to recognize the number, the prompt could be set up to return a "No Match" event. A prompt as follows may be played, for example:

> "I'm sorry, I did not understand you."

At this point, the prompt is repeated, but instead of the full length version as played the first time, a second, shorter, version of the prompt is played. For example:

> "What is your seven-digit account number?"

If the system still cannot understand the caller, a second version of the "No Match" prompt might be played. For example:

> "I'm sorry. I still didn't understand you."

A third version of the main prompt might be, as follows:

> "Please speak slowly and clearly, and say all seven digits of your account number."

And so on.

To use different versions of a prompt, each version to be used is designated as a different *level* of the prompt. Dialog Designer automatically creates the first level of a prompt when the prompt is created, but additional levels of a prompt must be created. So, for example, to use three different versions of a prompt, two new prompt levels must be added. Prompt levels are added within the **Prompt Main** tab of the Prompt File Editor. See Prompt File Editor—Prompt Main Tab.

Dialog Designer displays each prompt level in its own page in the Prompt File Editor. When a level is added, Dialog Designer adds a new tab to the Prompt File Editor. Dialog Designer displays and labels the tab according to what number the prompt level is, **1st**, **2nd**, **3rd**, and so on (up to 25).

The definition and settings for each prompt level are independent of all other prompt levels. That is, each level of the prompt can use its own segments, conditions, and SSML controls, without regard to what any other level of the prompt does. For more information about defining prompts in levels, see Prompt File Editor—Level Tabs.

About Play Order for Prompt Levels

In Dialog Designer applications, the order in which different versions of prompts are played, can also be controlled. Prompt levels allow for different versions of a prompt when a prompt is repeated for a caller. See Understanding Prompt Levels.

Depending on where the prompt is placed, the **Play Order** settings affect the way prompt levels are selected differently:

- If the prompt is used at the topmost level of a node, the only **Play Order** setting that affects the order or selection of the prompt level is **Random**. In this case, each time the call flow enters the node, Dialog Designer selects which prompt level to use on a random basis.

  The other **Play Order** settings have no effect on these prompts.

- If the prompt is used at any other level within a node, the selected **Play Order** option determines how prompt levels are used:

  – **Standard** - This option uses the number value of the level to determine when this prompt gets played. When the system reaches the highest numbered level, it uses that level for all following repeats.

    For example, suppose you want the system to use the first level of a prompt twice before the system uses the second version of the prompt. In other words, you want the second version of the prompt to *not* play until the third time the prompt is played.

    In this case, add one new level for the prompt, but instead of entering **2** in the **Please enter the attempt count** field of the **Prompt Count** dialog box, enter **3**. This entry causes Dialog Designer to display **3rd** in the level tab for this level.

    At run time, then, the system plays the **1st** prompt level before it plays the **3rd** level.

  – **First** - This option causes only the **1st** or lowest numbered prompt level to be played, no matter how many times the prompt is repeated. The system ignores all prompt levels other than **1st**. If there is no **1st** level, the system plays the lowest numbered prompt level.

  – **Random** - This option selects the level to play on a random basis, so the order in which the levels are played cannot be determined.

  – **Sequential** - This option plays each level once, in order from the lowest to the highest numbered level. When the system reaches the highest numbered level, it uses that level for all following repeats.

    For example, consider three prompt levels numbered **1st**, **4th**, and **5th**. With this option, the system plays the **1st** level on the first pass. On the second pass, the system plays the **4th** level. On the third and all following passes, the system plays the **5th** level.

Although it is unlikely that a prompt will need to be repeated more than three or four times, the following example describes how **Play Order** options work. For the sake of this example, assume that the prompt is supposed to repeat indefinitely. Assume also that there are four prompt levels, numbered **1st**, **3rd**, **5th**, and **6th**. The following table shows how these four prompt levels would be presented for each **Play Order** option:

Play Order option:	How levels would be presented:
Standard	**1st**, **1st**, **3rd**, **3rd**, **5th**, **6th**, **6th**, **6th**, **6th**...
First	**1st**, **1st**, **1st**, **1st**, **1st**, **1st**...
Random[1]	**5th**, **3rd**, **6th**, **3rd**, **1st**, **1st**, **1st**, **5th**, **6th**...
Sequential	**1st**, **3rd**, **5th**, **6th**, **6th**, **6th**, **6th**...

1. The order for this option cannot be predetermined, because the levels are presented in random order. The order presented in this example is only one possibility.

About Conditions in Prompts

To vary or change a prompt, depending on the situation, **Condition** items can be used to accomplish this within a single prompt.

For example, suppose the application that you are creating plays a different greeting to the caller based on business hours. The built-in time and date variables can be used to return the current day and time. Use a **Condition-If** item combination to test whether the call is coming into the system during business hours or otherwise. If the call is coming in during business hours, Dialog Designer plays the prompt on that branch of the condition tree. Otherwise, Dialog Designer plays the default prompt. Nested **If-Else** conditions can also be used, if needed.

For more information about using **Condition** items in prompts, see:

- If
- Else

About SSML Controls in Prompts

For the TTS-based prompt segments, **Text Variable** items and **TTS** items, SSML controls are used to refine the way the TTS engine renders the text.

Note:
The TTS engine must be SSML-compliant for these controls to work. Dialog Designer uses the Microsoft SDK TTS engine which is *not* SSML-compliant; these controls are ignored during testing.

Using SSML controls allows for the following functionality in prompts:

- Insert or eliminate breaks, or pauses, in the flow of the speech. See Break.

- Increase or decrease the amount of emphasis on words or phrases. See Emphasis.
- Use markers to detect barge-in during TTS playback. See Mark.
- Vary and control the pitch, rate, duration, and volume. See Prosody.
- Determine the way in which a word or phrase is interpreted. See Say As.
- Set the age and gender of the synthesized voice used for text rendering. See Voice.

About Transitional Audio Prompts

Transitional audio prompts are specialized prompts. They are similar to other prompts in many respects but different in significant ways. Usually, transitional audio prompts are used during transitions, for example, when data is being retrieved and the caller is waiting for a further response from the system. They are also usually used during bridge call transfers, while the caller is waiting for the transfer to complete.

Often, transitional audio prompts consist of a music selection that plays while the action is being completed.

For more details, see the following sections:

- Creating Transitional Audio Prompts
- Differences in Transitional Audio Prompts
- Using Transitional Audio Prompts

Creating Transitional Audio Prompts

A transitional audio prompt must be defined as such during prompt creation. You cannot take a prompt file that was not created as a transitional audio prompt and later convert it to be one.

To define a prompt as a transitional audio prompt, you must select the check box labeled **Transitional Audio prompt (allows only a single phrase)** in the prompt file wizard.

See Using the Prompt File Wizard to Create Prompt Files.

Differences in Transitional Audio Prompts

Transitional audio prompts differ from other prompts in several significant ways. With transitional audio prompts:

- With transitional audio prompts, prompt attributes on the **Prompt Main** tab cannot be changed. Level properties, as shown in the **Avaya Properties** view, can be changed however. These level properties include the **Timeout**, **Time Unit**, and **Barge In** properties, as discussed in the table in Prompt File Editor—Prompt Main Tab.
- Prompt levels cannot be added or deleted.

- The level (**1st**) tab palette offers only one type of prompt segment: **Phrase**. Also, there are no **SSML** items.
- There can only be one phrase as the prompt, unless **Condition** items are used. Even then, only one phrase per **If** or **Else** item can be used.

  A phrase file in the format of an audio (*.wav or *.au) file does not have to be speech. It can consist of a musical selection.

Using Transitional Audio Prompts

Transitional audio prompts can be used in the following two places:

- The **Property** form item - This usage is typical when the system requires more than a couple seconds to process an action or request by the caller. The idea is that the caller should never have the impression that nothing is happening. Some sort of audio cue, like music for example, can indicate to the caller that the system is processing data and the caller should stay on the line.

  See Property.
- The **Bridged Transfer** application item - This usage is typical when the system requires more than a couple seconds to complete a call transfer. The idea is that the caller needs to know that the call has not been disconnected, but rather that the system is working to complete the call transfer.

Note:
Any audio prompts in, or after a transfer node, will not get played before the call has been transferred. Putting the audio prompt as a transitional prompt when transitioning to the node that does the transfer ensures that the prompt is played before the transfer happens.

See the **Transfer Audio** prompt entry in Bridged Transfer.

Using the Prompt File Wizard to Create Prompt Files

To create a prompt file, use the Prompt File Wizard. Access the wizard using one of the following methodologies:

- From the **File** menu select **New** > **Prompt File**.
- On the main toolbar, click the icon for the prompt file wizard ().
- Right-click anywhere in the Navigator view, then select **New** > **Prompt File**.

Dialog Designer displays the Prompt File Wizard to create a prompt. To create the prompt, see the Create a Prompt Page section.

After creating a prompt file, to assemble prompt segments or to set other properties for the prompt file, use the Prompt File Editor. See Using the Prompt File Editor.

Create a Prompt Page

To create the prompt, complete the following fields in the Prompt File Wizard:

Creating a Prompt within the Prompt File Wizard

Fields	Description	Required?
Available Projects	Select a project to which the prompt file will be assigned.	Yes
File Name	Name identifying the prompt that intuitively describes the content of the prompt file. For example, a prompt that asks the caller for an account number might be named **AcctNmbr**. Dialog Designer automatically adds the **.prompt** extension to the file name when the prompt file is created. For transitional audio prompts, Dialog Designer adds a **.audioprompt** extension. **Note:** Prompt file names must follow Java naming conventions (see Naming Java Components).	No, but recommended. If a name is not entered, a generic name is assigned by Dialog Designer.
Transitional Audio prompt (allows only a single phrase)	To make the prompt file a transitional audio prompt, select this check box. Transitional audio prompts can be used in: ● the **Property** form item See Property. ● the **Bridged Transfer** application item See the **Transfer Audio** prompt entry in Bridged Transfer. For more information on transitional audio prompts, see About Transitional Audio Prompts.	No

Using the Prompt File Editor

Use the Prompt File Editor to define and modify prompt files once they are created. You can use the Prompt File Editor to:

- Set conditions that govern when different prompt segments are used. See About Conditions in Prompts.
- Add SSML controls to TTS segments. See About SSML Controls in Prompts.
- Define the behavior of transitional audio prompts. See About Transitional Audio Prompts.
- Set barge-in and play order attributes. Add, edit, and delete prompt levels. See Prompt File Editor—Prompt Main Tab.
- Assemble one or more prompt segments to make a complete prompt or announcement. See Building Prompts.

For more details on accessing the Prompt File Editor and the functional tab associated with it, see the following sections:

- Accessing the Prompt File Editor
- Prompt File Editor—Level Tabs
- Prompt File Editor—Prompt Main Tab

Tip:
To set the text that is synthesized for a TTS segment in the Prompt Editor, select the TTS segment and set the **TTS Text** property in the Properties view, or alternately click twice, slowly (not a double-click) on the TTS segment in the editor and the text can be edited in line.

Accessing the Prompt File Editor

To access the Prompt File Editor, use one of the following methodologies:

- Create a prompt file with the prompt file wizard. The prompt file wizard opens the Prompt File Editor automatically upon clicking **Finish** when creating a prompt file.

  To create a prompt file with the prompt file wizard, see Using the Prompt File Wizard to Create Prompt Files.
- In the Navigator view, double-click the ***.prompt** or ***.audioprompt** file you want to edit.
- For any prompt file that is already open in the Prompt File Editor workspace but not currently showing in the active workspace, click the Prompt File Editor tab for that prompt file.

Prompt File Editor—Level Tabs

When using multiple versions of a prompt, a *prompt level* for each version to be used must be created. Dialog Designer displays the definition and settings for each prompt level in its own tab in the Prompt File Editor. These tabs are located at the bottom of the workspace area when the Prompt File Editor is open.

Dialog Designer displays prompt levels in numeric order, labeled **1st**, **2nd**, **3rd**, and so on (up to 25). See Understanding Prompt Levels.

Use the level tabs to:

- Assemble prompt segments into prompts. See Building Prompts.
- Use conditions to determine when various prompt segments are played. See About Conditions in Prompts.
- Use SSML controls to regulate the way TTS content is rendered. See About SSML Controls in Prompts.

Each level tab has its own palette with the following options. For detailed information about each of the following options, click the links.

- **Segment** items
  - TTS
  - Phrase Variable
  - Text Variable
  - Audio Variable
  - Phrase Variable
  - Expression
- **Condition** items
  - If
  - Else
- **SSML** items
  - Break
  - Emphasis
  - Mark
  - Prosody
  - Say As
  - Voice

Prompt File Editor—Prompt Main Tab

Use the **Prompt Main** tab of the Prompt File Editor to:

- Set barge-in behavior for the prompt.
- Determine the way in which prompt levels are presented.
- Add, edit, and delete prompt levels. For more information about prompt levels, see Understanding Prompt Levels.

Tip:
To set the text that is synthesized for a TTS segment in the Prompt Editor, select the TTS segment and set the **TTS Text** property in the Properties view, or alternately click twice, slowly (not a double-click) on the TTS segment in the editor and the text can be edited in line.

This tab is located at the bottom of the workspace area when the Prompt File Editor is open.

The **Prompt Main** tab has the following fields and options:

Prompt Main Tab Options/Fields

Option/Field	Action/Comments	Required?
Prompt Attributes		
Display Name	(Display only) Displays the name of the prompt file, as assigned when the file was created.	N/A
Bargein Type	Type of action that triggers barge-in, if enabled for the prompt. Options are: • **speech** - Any spoken response triggers barge-in and stops the prompt. This is the default. • **hotword** - Only a complete match of an active grammar triggers barge-in and stops the prompt. To edit the properties of a prompt level, click to select the desired level. When you select a level, you can then edit the level properties in the **Avaya Properties** view. These properties include: • **Timeout** - Number of seconds or milliseconds that the system must wait for a response from the caller, after the prompt is finished playing, and before the system throws a No Input event. • **Barge In** - Enable (true) or disable (false) barge-in for the specified prompt level. By default, barge-in is enabled.	Yes

Prompt Main Tab Options/Fields (continued)

Option/Field	Action/Comments	Required?
Play Order	Determines the order in which prompt levels are played when the prompt is repeated. Options include: • **standard** (default) • **first** • **random** • **sequential** See About Play Order for Prompt Levels.	Yes
Language	(Display only) Displays the default language assigned to the application.	N/A
Prompt Levels		
[display pane]	(Display only) Displays all the levels currently defined for the prompt.	N/A
Edit	Click to edit the prompt for this level. To activate this button, select a prompt level in the **Prompt Level** display pane. When you click this button, Dialog Designer directs its focus to the level tab for the selected level (a maximum of 25). See Understanding Prompt Levels.	N/A
Add	Click to add a prompt level. When you click this button, Dialog Designer displays the **Prompt Count** dialog box. This dialog box has fields and options to: • Enter the attempt count number for this level (**Please enter the attempt count** field). • Select another level to copy as the starting point for this level (**Copy from existing prompt**). The drop-down list displays all existing prompt levels. Alternately, you can select not to copy any existing level (**none**).	N/A
Delete	Click to delete a selected prompt level. To delete the level, first select the prompt level and then click this button.	N/A

Exporting and Importing Prompt Files

In Dialog Designer, prompts from one speech application can be reused in another speech application projects. This is useful when a common set of prompts are used in two or more applications, or when you have a speech module that uses the same prompts as the parent application.

See the following sections for more information:

- Exporting a Prompts Zip File
- Importing a Prompts Zip File
- Importing a Delimited Prompt Data File

Exporting a Prompts Zip File

To export a Prompts Zip File, so they can be reused in more than one speech application project, use the Export Prompts Wizard.

To access the Export Prompts Wizard, perform one of the following methodologies:

- From the **File** menu, select **Export...**
- Right-click anywhere in the **Navigator** view, then select **Export...**

> **Important:**
> Be careful when exporting prompt files. Prompts typically rely on phrases, grammars, and other project resources that may not exist in the project you are exporting the prompts to. If you import prompt files that rely on other resources into other projects, Dialog Designer generates errors to let you know what resources are missing. You must then either export the missing resources for the other project or redirect the imported prompts to make use of other resources in that other project.

Select the following appropriate options within the Prompts Export Wizard.

Prompts Export Wizard

Option	Description
Select Page	
Select an export destination	Select **Avaya Speech Development > Prompts Zip File**, then click **Next**.

Prompts Export Wizard (continued)

Option	Description
Phrases Page	
Available Projects	Select the project from which to export prompts.
Available Languages	Select the languages for which you want to export the prompts. This field lists only those languages for which prompts exist in the selected project.
Zip file	Perform one of the following actions: ● Enter the full path to and filename to be assigned to the exported .zip file. ● Use **Browse** to locate and destination of where the .zip file will be exported.
Options	Optionally, select the **Include directory structure** checkbox to include directory path information within the .zip file.

Importing a Prompts Zip File

To import a Prompts Zip File into a speech application project, a file must first be exported from another project. The exported file must then be imported into the project where they will be used.

Important:
Be careful when importing prompt files. Prompts typically rely on phrases, grammars, and other project resources that may not exist in the project where they are being imported. If you import prompt files that rely on other resources, Dialog Designer generates errors. Either import the missing resources or redirect the imported prompts to make use of other resources.

Note:
The following procedure applies only to prompt files that were exported to a .zip file using the Export wizard in Dialog Designer.

To access the Import Prompts wizard, use on of the following methodologies:

- From the **File** menu, select **Import...**
- Right-click anywhere in the **Navigator** view, then select **Import...**

Select the following appropriate options within the Prompts Import Wizard.

Prompts File Import Wizard

Option	Description
Select Page	
Select an import destination	Select **Avaya Speech Development > Prompts Zip File**, then click **Next**.
Phrases Page	
Phrase zip file	Perform one of the following actions: • Enter the full path to and name of the .zip file to be imported. • Use **Browse** to locate and select the .zip file to be imported.
Available Projects	Select the project for which the prompts will be imported.
Overwrite existing resources without warning	To overwrite any existing prompts files with the same names in the project, without warning, select this check box.

Importing a Delimited Prompt Data File

The Import Prompt Data option allows you to import new prompt data from a delimited text file to update existing prompts, and/or add new prompts, to your speech application project. This capability can be extremely helpful to batch create or update prompts that play pre-recorded audio.

Note:
Multiple prompt levels and multiple simultaneous languages are not supported by the Import Prompt Data option

Before importing a prompt data file, you need to first create one—a delimited prompt data text file. The text file must have the following format for each row:

```
<prompt>,<phraseset>:<phrase>,<phraseset>:<phrase>, ... n phrases per prompt
```

and/or

```
<prompt>,<phraseset>:<phrase>,<standalone_phrase>, ... n phrases per prompt
```

Each prompt can be made up *n* number of *phraseset:phrase* pairs and/or *standalone* phrases, or both. There is no limit on how many phrases can make up a single prompt.

The delimiter used in the text file can be a Tab, or another consistently used character in your file that does not conflict with the data content; for example a pipe symbol (“ | “) or comma.

To import a delimited prompt data file:

1. Create a delimited prompt data text file, as described in the previous discussion.
2. Within the *Dialog Designer Navigator* panel, select the speech application project to import the prompt data to.
3. From the *Main Menu* bar, select **File > Import**. The *Import selection* dialog is shown.
4. Select **Avaya Speech Development > Prompt Data**, then click **Next**.
5. The *Import Prompt Data* dialog is displayed. Click **Browse** to navigate to your delimited prompt data text file to be imported.
6. In the *Target Language* panel, select a **Project** and **Language** which will be updated based on the data within the prompt data file.
7. In the *Options* panel, define the delimiter used in your prompt data file. Select either the **Tab** option, or **Other** and indicate the delimiter character.
8. When done, click **Next**.
9. The **Import Prompt Data—Import Preview** dialog is shown, showing the anticipated import results before actually performing the import.
10. Review the preview, and if it all seems correct, click **Finish** (of **Back** if needed to rectify anything).

Building Prompts

To build a prompt, one or more prompt segments must be assembles in the order that they are expected to play. Any combination of prompt segment types can be used. For more information on prompt segment types, see Prompt Segment Types.

For more details on actually building a prompt, see the following sections:

- Before Building the Prompt
- Building the Prompt

Before Building the Prompt

Before a prompt can be built, all required prompt segments must be created.

For instance, suppose you want to create a prompt that reads the address of a caller from a variable. The prompt then asks if this address is still correct. What you want the caller to hear is: “According to our records, you live at [*address record*]. Is this correct?” This prompt requires two phrase segments and one text variable segment:

- Phrase segment: “According to our records, you live at”
- Text variable segment: The value returned by the database query, to be rendered using TTS
- Phrase segment: “Is this correct?”

Before you can build this prompt, you must use a database operation to read the address from the database. Then you must assign the value returned by the database operation to a variable, which you will later assign to the text variable in the Prompt File Editor.

You must also create the required phrases must be created. For information about creating phrases, see About Custom Phrases.

Building the Prompt

When all required prompt segments are created, the prompt can be assembled. To build a prompt:

1. Click an item in the Level tab palette.
2. Click in the main workspace area where you want the item to go.
3. Continue doing this until you have all the prompt segments, conditions, and SSML controls you need to make the prompt behave the way you want.
4. When all segments are available, in the palette for the Level tab of the prompt, click **Phrase**.
5. Click in the main workspace.

   Dialog Designer displays a **Phrase** item in the workspace.
6. In the **Avaya Properties** view, assign the appropriate phrase file to the **Phrase** item.

   In this case, it would be the phrase that says: “According to our records, you live at”.
7. In the level tab palette, click **Text Variable**.
8. In the main workspace, click somewhere below the **Phrase** item.

   Dialog Designer displays a **Text Variable** item in the workspace, immediately below the **Phrase** item.
9. In the **Avaya Properties** view, assign to the **Text Variable** item the variable that contains the address returned by the database operation.
10. In the level tab palette, click **Phrase**.
11. In the main workspace, click somewhere below the **Text Variable** item.

    Dialog Designer displays a **Phrase** item in the workspace.
12. In the **Avaya Properties** view, assign the appropriate phrase file to the **Phrase** item.

    In this case, the phrase would be: “Is this correct?”

Chapter 9: Working with Variables

The concept and use of variables should be familiar to anyone knowledgeable about programming concepts in general. For this reason, this documentation does not cover the basic concepts and uses of variables.

This chapter describes the particular aspects of working with variables in Avaya Dialog Designer. It contains the following sections:

- About Variables in Dialog Designer
- Using the Variable Editor
- Employing Variables in Applications
- Variables Generated by Dialog Designer

> **Important:**
> Variables, as described in this chapter are only used with speech applications. Call control application variables are incorporated within a CCXML file and are nothing like speech application variables.

About Variables in Dialog Designer

Variables can be used in Dialog Designer applications much like they are used in any other application. Because of the specialized nature of speech applications, there are special types and uses of variables in Dialog Designer.

The following sections provide an overview of the special types and uses of variables in Dialog Designer

- Types of Variables in Dialog Designer
- Use of Variables in Dialog Designer
- About Passing Variable Values

Types of Variables in Dialog Designer

Dialog Designer provides several different types of variables, as follows:

- **Simple variables** – Simple variables hold only one variable value at a time. This type is, perhaps, the most widely known type of variable in programming practice.

- **Complex variables** – Complex variables hold multiple values within a single variable. Dialog Designer accomplishes this with the use of associated *fields*. To be functional, a complex variable must have one or more variable fields assigned to it.

Note:
Another way that variables can hold multiple values in Dialog Designer is with the use of *collections*. This result happens in cases where a variable, either simple or complex, returns more than one value for the variable or variable field. Examples of this include:

- An **N Best** property that causes an operation to return several values for a single variable
- A database query that returns more than one record
- A Web service operation that returns an array

Dialog Designer has three basic ways of creating variables:

- **System variables** – Some variables are automatically built-in to and created with each speech project you create in Dialog Designer. These variables are known as *system variables*. These read-only variables are used for common operations in speech applications.

  For a complete list of the system variables and information about how to use them, see System Variables.

- **Generated variables** – For some palette options, Dialog Designer automatically generates variables. Most of these variables and their associated fields are read-only, but in a few, cases, custom fields can be added.

  Dialog Designer-generated variables can be either simple or complex variables, depending on what palette options they are associated with. For a list of the palette options that automatically create variables, see Variables Generated by Dialog Designer.

- **Custom variables** – Using the Variable Editor, custom variables—either simple or complex—can be created.

Note:
A complex variable requires at least one field to be functional. So, when you create a complex variable, Dialog Designer automatically creates a field and attaches the field to the variable.

For more information about creating variables, see Creating Variables.

Use of Variables in Dialog Designer

Variables in Dialog Designer applications can be used, among other things, to:

- Contain data for processing or manipulation
- Hold information about audio file locations

- Pass data to and from other modules or applications
- Receive data from or pass data to other connected systems, such as the Avaya Interaction Center (IC) or computer telephony integration (CTI) systems
- Variables can be set to be null

About Passing Variable Values

Variables in Dialog Designer are always local to an application or project. For example, if you are using a Dialog Designer speech project as a module in another application, the application cannot “see” the variables in that module. Any information that passes between them must be passed as input or output parameters.

Dialog Designer provides several ways for applications to pass variable values back and forth, as described in the following sections:

- Between Applications and Modules
- Between Applications and VoiceXML Objects
- Between Applications and IC Systems
- Between Applications and CTI Systems
- Between Speech and Call Control Applications

Between Applications and Modules

Variable values can be passed between a Dialog Designer speech application project and another module. This module can be another Dialog Designer-created speech project. Alternately, it can be a module created with a different tool, such as the OSDMs created by Nuance software.

To pass values between a parent Dialog Designer speech application and another module, you must use the **Input Parameter** and **Output Parameter** items. For more information about using these items to pass variable values, see:

- Input Parameter
- Output Parameter

Between Applications and VoiceXML Objects

In Dialog Designer, VoiceXML objects can be used to extend the capabilities of your speech application beyond the normal capabilities of VoiceXML. These objects can consist of code you write yourself or that you have obtained from someone else.

To pass values between a Dialog Designer speech application and a VoiceXML object, you must use the **Object**, **Object Input**, and **Object Output** items. For more information about passing variable values between Dialog Designer speech projects and VoiceXML objects, see:

- Object
- Object Input
- Object Output

Between Applications and IC Systems

Dialog Designer has a built-in connector that you can use to establish contact with and pass values between an Avaya IVR system and an Avaya Interaction Center (IC) system (such as IC 6.1.3-7.1). With this connector, you can leverage the strengths of both systems within a single application.

See About the IC Connector.

Between Applications and CTI Systems

Dialog Designer has a built-in connector that you can use to establish contact with and pass values to and from a computer telephony integration (CTI) system (such as AES 3.1 or AvayaCT 1.3). With this connector, you can extend the capabilities of Dialog Designer applications beyond the normal limitations of VoiceXML Version 2.0-compliant applications.

Between Speech and Call Control Applications

From CCXML to speech application you use the namelist in the **dialogprepare** or **dialogstart** tag to pass values to a speech application from a ccxml application. In the speech application you would use the Capture Expression to store that into a Dialog Designer variable.

If the namelist in CCXML is **namelist="myvar myothervar" />** then the expression in the speech application to access the passed in value is **session.connection.ccxml.values.myvar** and **session.connection.ccxml.values.myothervar**.

To pass data back to a CCXML application use add an output parameter to the return. The returned values show up in CCXML in **dialog.exit** event **event.values.***.

So if **dialogresult** and **numbertocall** is returned in the transition for the **dialog.exit** event, the return values can be accessed as:

```
<transition event="dialog.exit" name="evt" state="in_dialog">
  <log expr="' dialogresult : ' + evt.values.dialogresult"/>
  <log expr="' numbertocall : ' + evt.values.numbertocall"/> ...
  ... ...
```

Note:
Only simple strings can be passed back and forth. Objects are not allowed.

Using the Variable Editor

Using the Variable Editor, the developer is provided with options described in the following sections:

- Accessing the Variable Editor
- Creating Variables
- Deleting Variables

Accessing the Variable Editor

When the Variable Editor is accessed, all variables that currently exist within the speech application project are displayed. Additional editing functions are also available.

To access the Variable Editor:

1. In the Navigator view, open the **flow** subdirectory of the project directory.
2. Double-click **project.variables**. Dialog Designer displays the Variable Editor in the main workspace.

Simple variables, complex variables, and variable fields can be identified by their differing icons. These icons, and their labels are shown in the palette at the left side of the workspace.

To view the fields associated with a particular complex variable, click the **+** symbol to the left of a variable icon and name.

Tip:
To edit the name of any variable (or field) in the Variables Editor, select the variable or field and set the **Name** property in the properties view, or alternately click twice, slowly (not a double-click) on the variable (or field) and the name can be edited in line.

Creating Variables

To create a custom variable:

1. Access the Variable Editor, if not already open. To open the Variable Editor, see Accessing the Variable Editor.
2. Click the palette option for the type of variable you want to create, either **Simple Variable** or **Complex Variable**.

3. Click inside the main workspace area.

   Dialog Designer displays the new variable and automatically assigns it a temporary default name.

4. Optionally, enter a descriptive name in the **Name** field of the Avaya Properties view.

CAUTION:
Do not use double-byte or extended ASCII characters when naming variables.

Note:
When creating a complex variable, at least one **Field** item must be added to it before it is functional. Use this same procedure to add fields to complex variables.

Once the variable is created, the variable properties can be set in the Avaya Properties view.

See Simple Variable or Complex Variable.

Deleting Variables

To delete a variable or variable field:

1. Access the Variable Editor if not already open. See Accessing the Variable Editor.
2. Select the variable or variable field you want to delete. Use shift-click to select multiple adjacent variables or [Ctrl]-click to select multiple nonadjacent variables.
3. Press **Delete**.

Employing Variables in Applications

Dialog Designer uses variables in applications in the following ways:

- Use the palette options and items in many nodes to employ variables. See Using Node Items to Employ Variables.
- Use the Prompt File Editor to employ audio or text variables as prompt segments within prompts. See Using the Prompt File Editor to Employ Variables.
- Employ call session variables programmatically. See Programmatically Accessing Call Session Variables.

Using Node Items to Employ Variables

Many nodes have palette options in which you can make use of variables in your Dialog Designer speech projects. To offer a few examples, you can:

- Use an **Operation** palette option in the Data node to assign values to variables or to perform calculations or other manipulations on variable values.
- Use a database operation to retrieve a telephone number from a database, and then assign that number to a variable. You can then use that variable as the transfer destination variable in a call transfer.
- Include application performance and call information data in variables within a Report item. This data can be sent to an IVR system such as Avaya Voice Portal. There you can use it to prepare reports and analyze application data.

Other nodes have options in which you can make use of the system variables to set branching conditions, for instance. You can also use system variables pass variable values to other application modules.

In each case, the variable must exist before you can use it in a node item. If the variable does not already exist as a system variable or a Dialog Designer-generated variable, you must create a custom variable to do the job.

In all cases, palette options and items in which you can employ variables or variable fields are documented in the individual descriptions of those options and items. See Nodes and Palette Options.

Using the Prompt File Editor to Employ Variables

The Prompt File Editor offers two ways to render, or play, a variable value depending on the type of variable being rendered.

See the following sections:

- About Rendering Audio Variables
- About Rendering Text Variables

About Rendering Audio Variables

Audio variables render variable information into audio outputs by parsing the variable content and then concatenating prerecorded audio files to play back the content.

To create a variable or variable field that can be used as an audio variable, you must ensure that the value of the variable matches a format that the localization bundle expects. For example, all localization bundles expect variable data that is to be rendered as a date to be in the format *YYYYMMDD* or *YYYY-MM-DD*, where:

- *YYYY* is the four-digit year
- *MM* is the two-digit month
- *DD* is the two-digit day

See the following sections:

- Variable Formats in Localization Bundles
- Audio Variable

About Rendering Text Variables

Text variables render variable information of various formats as audible output using Text-to-Speech speech synthesis technology.

Almost any variable can be used as a text variable, though you must exercise some caution in selecting a variable to be rendered with TTS. Basically, what Dialog Designer does is take the value of the variable and pass it along, in string format, to the TTS server for rendering. Therefore, if you use a variable whose contents consist of a Java object reference, for instance, you might get unexpected results when the TTS server renders the content as synthesized speech.

Be aware also that not all TTS servers render the same variable values in the same way. One TTS server, for instance, might render the date 05-25-2005 as “May twenty-fifth, two thousand five.” A different TTS server might render the same date as “oh five dash twenty-five dash two thousand and five.” This fact means that, to use text variables most effectively, you must know the formats and functions of your TTS server.

See Text Variable.

Programmatically Accessing Call Session Variables

Call session variables (simple, complex, or system) can be accessed or updated in any of the generated Java code or developer-defined code that has access to a **com.avaya.sce.runtimecommon.SCESession** object. The **SCESession** object stores Dialog Designer-generated variables and can also store any variables defined by the application developer.

In a typical Dialog Designer project, most application developers would programmatically access session variables in one or two places—via the Servlet Node or other nodes, such as a Form, Menu, or Data node, for example.

This section describes ways to programmatically access call session variables within Dialog Designer applications. It contains the following sections:

- General Approach to Accessing Variables
- Getting Call Session Variable Values

- Setting Call Session Variable Values
- About IProjectVariables

General Approach to Accessing Variables

The general approach of accessing call session variables—either simple or complex— is to add code to a node.

For a Servlet Node, override the following method:

public void servletImplementation(SCESession mySession)

For an “Other” Node (for example, Form, Menu, Data, etc.), override the following method:

public void requestBegin(SCESession mySession)

Getting Call Session Variable Values

To get call session variables programmatically, the methodology is a little different depending on the type of variable: a simple or complex variable.

See the following sections:

- Getting Simple Variables
- Getting Complex Variables

Refer to the *Dialog Designer Programmers Guide* in the online help for the full set of API for accessing variables and their values.

Getting Simple Variables

The following code will get the value of the simple variable “weather” as a Java String object.

Note:
This code assumes that a simple variable “weather” has been defined in the project Variable Editor. See About IProjectVariables.

```
IVariable variableRef = mySession.getVariable(IProjectVariables.WEATHER);
IVariableField variable = variableRef.getSimpleVariable();
String weather = variable.getStringValue();
```

A short-hand way of writing the same code is:

```
IVariableField variable = mySession.getVariableField(IProjectVariables.WEATHER);
String weather = variable.getStringValue();
```

Getting Complex Variables

The following code will get the value of the complex variable field “date:year” as an integer.

```
IVariable variableRef = mySession.getVariable(IProjectVariables.DATE);
IComplexVariable complexVariable = variableRef.getComplexVariable();
IVariableField field =
complexVariable.getField(IProjectVariables.DATE_FIELD_YEAR);
int year = field.getIntValue();
```

A short-hand way of writing the same code is:

```
IVariableField field = mySession.getVariableField(IProjectVariables.DATE,
IProjectVariables.DATE_FIELD_YEAR);
int year = field.getIntValue();
```

Setting Call Session Variable Values

To set call session variables programmatically, the methodology is a little different depending on the type of variable: a simple or complex variable.

See the following sections:

- Setting Simple Variables
- Setting Complex Variables

Refer to the *Dialog Designer Programmers Guide* in the online help for the full set of API for accessing variables and their values.

Setting Simple Variables

The following code sets the value of the simple variable “weather” to a Java String object.

Note:
This code assumes that a simple variable “weather” has been defined in the project Variable Editor. This code uses the short-hand methods to access variables.

```
String weather = "It is cold outside!";
IVariableField variable = mySession.getVariableField(IProjectVariables.WEATHER);
variable.setValue(weather);
```

Setting Complex Variables

To set the value of a complex variable field, code format to obtain an IVariableField reference for a complex variable field.

SCESession.getVariableField(String *variableName*, String *fieldName*)

About IProjectVariables

Dialog Designer generates a class, ***IProjectVariables***, which defines the names of all project variables and their fields as constant Java strings. Dialog Designer will automatically generate this class using all of the variables defined in the **flow/project.variables** file (Variable Editor).

Application developers are encouraged to define their own constants for any variables that are added to the session programmatically. When editing the IProjectVariables class, be sure to make changes *outside* of the code generation tags (***//{{START:*** etc.) or changes may be overwritten by the code generator.

Developers are encouraged to reference variables using the constants defined in IProjectVariables because it reduces the possibility of application bugs when variables are renamed. For example, a developer has defined the variable “weather” in the project variables and then elsewhere in the application written the code:

IVariableField variable = mySession.getVariableField(“weather”);

A second developer modifies the application and renames “weather” to “theWeather” in the project variables. Now the code “**mySession.getVariableField(“weather”)**” results in a runtime error because the variable “weather” is no longer a project variable.

If the developer originally wrote the following code:

“mySession.getVariableField(IProjectVariables.WEATHER)”,

then when a second developer renamed the project variable, there would be a Java compile error because the constant **“IProjectVariables.WEATHER”** would no longer exist; it would be changed to **“IProjectVariables.THE_WEATHER”**.

Variables Generated by Dialog Designer

There are several palette options and items in the Dialog Designer user interface which, when used, Dialog Designer automatically generates corresponding variables that have the same name. For example, if using a Blind Transfer node in an application, Dialog Designer automatically generates a variable for the node. This is done because the Blind Transfer node contains a Blind Transfer item. When a Blind Transfer item is used, Dialog Designer automatically generates a complex variable with the same name.

Dialog Designer generates an associated read-only variable for the following palette option items:

- Blind Transfer
- Bridged Transfer
- Call Info (CTI)
- Blind Call (CTI)

- Consultation Call (CTI)
- Dial (CTI)
- Input
- Input Parameter
- Module Output
- Object Output
- Record

In addition to these, if the Interaction Center (IC) connector is enabled for your application, Dialog Designer generates two complex variables: **vdu** and **vdu_cache**. Custom fields can be added to the **vdu** variable, but not the **vdu_cache** variable.

See About the IC Connector.

Chapter 10: Working with Grammars

The key to recognition of caller input is the use of grammars. In the context of IVR applications, a grammar is a predefined set of words or DTMF tones that a speech application uses to interpret and respond to caller inputs.

This chapter describes all aspects of creating and using grammars in Avaya Dialog Designer. It contains the following sections:

About Grammars in Dialog Designer

One of the basic functions of an interactive voice response (IVR) system is to collect and recognize input from the caller. This input is the form of either spoken voice responses, or touchtone key presses, known as dual tone multi-frequency (DTMF) input. The system uses this input to direct the flow of a call.

A grammar must be used anywhere in a speech application where caller input needs to be collected that the system can process. The exceptions to this rule are those nodes and node items in which you can use built-in DTMF or speech recognition.

See the following sections for information on using grammars:

Types of Grammars

Dialog Designer uses two types of grammars:

- *Voice* (speech) grammars are designed for automated speech recognition (ASR) servers for use with speech recognition and interpretation. These grammars are used in a speech application project where spoken responses need to be collected.

  To use voice grammars, an ASR server is required to process the input when deploying the application on an IVR system. Dialog Designer uses the Microsoft Speech SDK for testing and simulation. For run-time deployment, any SAPI-compliant ASR server can be used.

- *DTMF grammars* are designed to recognize combinations of one or more touchtone key presses on a telephone keypad. With many Dialog Designer nodes and items, DTMF grammars do not need to be defined if the system should respond to single key presses. In some node items, such as the Choice item used in the Menu node, single *and* multiple key presses can be defined within the item, without having to use a DTMF grammar.

  If the **Show AutoVon keys** option is checked within the Avaya Application Simulator Preferences, four more DTMF buttons are added next to the dial keypad for sending A, B, C, and D AutoVon tones for input.

  The term “multiple key presses,” in VoiceXML applications, does not mean multiple *consecutive* key presses, as it does in some other types of IVR systems. In VoiceXML applications, the term “multiple key presses” means that you have multiple *single* key presses available for the same menu choice.

  For example, in a Choice menu, if the following **#** and **7** DTMF key options were entered, the VoiceXML application may interpret the key presses in such a way that either a # key press *or* a **7** key press activates the menu choice.

Planning and Designing Grammars

You must plan carefully when designing your grammars. It is important to realize that callers do not always respond the way you might expect them to, and you must try to anticipate all the possible responses you might get to a particular prompt.

For example, even when you want to solicit a simple “yes” or “no” response from callers, you should realize that caller responses might include “yeah,” “OK,” “yep,” and “sure” in place of “yes.” Similarly, negative responses might include “nah,” “nope,” or “uh-uh” in place of “no.”

Using Multiple Grammars

It is very possible, even likely, that at some point in your speech application you may have overlapping grammars. Overlapping grammars occur when the system has two or more active grammars for which it can accept responses.

For example, there can be a grammar set up in the AppRoot node whose function is to listen for requests for help or to be directed to a live attendant. Such grammars are nearly always active. Even within the same node there can be multiple grammars, for example, when using the Menu node.

You can also use multiple grammars with a single field. For example, you might want to offer callers the option to respond with either a verbal response or a DTMF key press. In this case, you might want to create a grammar for the ASR response and a different grammar for the DTMF response. When used on the same field, either grammar can trigger the recognition. Good application design might even require you to create multiple ASR grammars for the same response field, especially when designing applications that use a natural language speech recognition approach.

Be aware when you have overlapping grammars in your application. Be careful also to ensure that you do not also have overlapping responses or entries within them.

Compatibility and Compliance

By default, Dialog Designer generates grammars that are compliant with the W3C Speech Recognition Grammar Specification (SRGS), version 1.0. This fact means that they can be used with any ASR server that can process SRGS-compliant grammars, including ASR servers and software produced by companies like IBM (WebSphere) and Nuance.

Note:

As is often the case with emerging technologies, the way different companies approach and interpret a new standard is somewhat different. This fact is evident in the way different companies have approached the use of the <tag> element in the SRGS Version 1.0 specification.

Be aware of these differences when creating grammars for your particular system. For more information about how your ASR server implements the SRGS Version 1.0 specification, see the documentation for your ASR server.

Important:

If you are using Voice Portal 3.0 and Nuance OSR, use only dynamic grammars. Nuance OSR does not static/external grammars well.

Basic Process of Using Grammars in Dialog Designer

To use a grammar in a speech application, multiple steps must be performed, as follows:

1. Create and define a grammar file. To create and define grammars, see:
   - Using the Grammar File Wizard to Create Grammar Files
   - Using the Grammar File Editor
2. Place a **Grammar** item in the node (sub)flow in the node editor at point where you need to collect and interpret speech input.

   See Grammar.
3. Use the **Avaya Properties** view to assign the grammar file to the **Grammar** item in the node editor.

Using the Grammar File Wizard to Create Grammar Files

A grammar is a predefined set of words or DTMF tones that a speech application uses to interpret and respond to caller inputs. To create a grammar, a grammar file must be created to hold the grammar entries.

To create a grammar file, use the Grammar File Wizard. See the following sections:

- Accessing the Grammar File Wizard
- New Grammar Page

After a grammar file is created, entries and settings to be used by the grammar must be defined. To define grammar entries, use the Grammar File Editor. See Using the Grammar File Editor.

Accessing the Grammar File Wizard

To access the Grammar File Wizard, use one of the following methodologies:

- From the **File** menu select **New** > **Grammar File**.
- On the main toolbar, click the New Grammar File icon ().
- Right-click anywhere in the Navigator view, then select **New** > **Grammar File**.

Dialog Designer displays the New Grammar Page of the Grammar File Wizard.

New Grammar Page

Configure the following fields and options within the New Grammar Page of the Grammar File.

Grammar File Wizard—New Grammar Page

Field or Option	Description
Available Projects	Select the project to which the grammar file will be used.
File Name	Name to identify the grammar. **Note:** Grammar names must follow Java naming conventions (see Naming Java Components). **CAUTION:** Do not use double-byte or extended ASCII characters when naming the grammar file.
Type	Select a **Type** of the grammar to be created. Possible types are: • **Static** - A grammar with a limited number of entries. The most common type of grammar, static grammars have a **.gram** assigned to their filenames. See Editing Static Grammars. • **Dynamic** - A Java class grammar that uses an API to build the elements that make up a grammar. At run-time, a dynamic grammar (the Java class) generates the appropriate grammar content dynamically, by retrieving values from a data source such as a database, an LDAP, or a Web service. When clicking **Finish** to create the grammar file, Dialog Designer automatically opens a Java class editor that is populated with sample code to help get started. Dynamic grammars have a **.gram-dyn** assigned to their filenames. See Editing the Dynamic Grammar Java Class. • **External** - An external grammar is one that is created, defined, and maintained outside of Dialog Designer. To use an external grammar, the name and path to the grammar file, and what domain host the grammar file resides on must all be known. External grammars have a **.gram-extrn** assigned to their filenames. See Editing External Grammar Access Properties.
Mode	Indicates the mode of the grammar. Options are: **DTMF** or **Voice**. **Note:** Only available for a **Static** Type.
Style	Indicates whether **Custom** or **Built-in** grammars will be used. Built-in grammars have a **.gram-builtin** extension assigned to their filenames. See Selecting Built-in Grammar Types. **Note:** Only available for a **Static** Type.

Grammar File Wizard—New Grammar Page (continued)

Field or Option	Description
Number of rows	For each caller response to be defined for recognition, a row must be accounted for in the grammar. Enter the number of rows anticipated for the grammar. See Editing Static Grammars. **Tip**: Rows are automatically added to the bottom of a row by pressing the [Return] key.
Number of columns	Each column in a static grammar table represents a potential system response to a caller utterance. Enter the number of columns anticipated for the grammar. See Editing Static Grammars.
Open file for editing when done	To specify for Dialog Designer to open the Grammar File Editor when finished creating the grammar file, select this option. See Using the Grammar File Editor. **Note:** Only available for **Static** or **External** Types.
Generate sample grammar	To specify for Dialog Designer to create a simple populated grammar as a starting point for help in defining the grammar, select this option.

When done, perform one of the following actions:

- To create the grammar file without defining the grammar entries (but which are required for a grammar to be useful in an application), click **Finish**.
- Dialog Designer creates the grammar file. If the **Open file for editing when done** option was checked when creating the grammar file, the Grammar File Editor is displayed. See Using the Grammar File Editor.
- To exit the Grammar File Wizard without creating the phrase file, click **Cancel**.

Using the Grammar File Editor

Use the Grammar File Editor to define entries for previously created grammar files (Using the Grammar File Wizard to Create Grammar Files). Grammar entries define a predefined set of words or DTMF tones that a speech application uses to interpret and respond to caller inputs.

After accessing the Grammar File Editor, the type of editor window launched in the workspace depends on the type of grammar being edited. See the following sections for more information:

- Accessing the Grammar File Editor
- Editing Static Grammars
- Editing the Dynamic Grammar Java Class
- Editing External Grammar Access Properties

Accessing the Grammar File Editor

To access the Grammar File Editor, use one of the following methodologies:

- Create a grammar file with the Grammar File Wizard (see Accessing the Grammar File Wizard). If the grammar **Type** selected is **Dynamic** or the check box labeled **Open file for editing when done** is selected when creating a grammar, Dialog Designer opens the Grammar File Editor automatically upon clicking **Finish** after creating a grammar.
- In the Navigator view, double-click the ***.gram*** file to edit. The exact extension on the grammar file depends on the type of grammar. For more information about grammar types and extensions, see Using the Grammar File Editor.
- For any grammar file that is already open in the Grammar File Editor workspace but not currently displayed in the active workspace, click the Grammar File Editor tab for that grammar file.

After accessing the Grammar File Editor, the type of editor window launched in the workspace depends on the type of grammar being edited. See the following sections for more information:

- Editing Static Grammars
- Editing the Dynamic Grammar Java Class
- Editing External Grammar Access Properties

Editing Static Grammars

The Grammar File Editor for a static grammar uses a tabular format to define the entries that comprise the grammar. Each column is a rule for reference-able column value items. For example, Column 1 and 2 can be renamed to something like “animal” and “color” by clicking on the column name.

When a grammar is placed on an input field, two additional fields are added (in addition to the standard utterance, confidence, input mode, value, and interpretation). So for example, “animal” can be set to “dog” and “color” can be set to “blue”.

When first accessing the Grammar File Editor for a static grammar, the table consists of only one active column and row. In effect, this means there is only one active cell, which is initially unpopulated.

If you selected the check box labeled **Generate sample grammar** in the grammar file wizard, the Grammar File Editor opens with three populated cells.

See the following sections for information on editing information in the Static Grammar File Editor:

- Adding (or Deleting) Rows to a Static Grammar Table
- Adding (or Deleting) Columns to a Static Grammar Table
- Using the TAG Option

- Setting Static Entry Properties

Adding (or Deleting) Rows to a Static Grammar Table

For each caller response that you want to want to define for recognition, you must add a row. For example, if you are offering a list of food items that a caller can order, you might want to list:

- Hamburger
- Cheeseburger
- French fries
- Onion rings
- Salad

To offer all these as options, at least one row entry needs to be created for each food item. In this example, consider offering multiple entries for food items, based on expected caller responses. For example, “hamburger” and “burger.”

To add rows to the static grammar table:

1. Click an active cell in the table.

   Note:
   You can identify rows that have been created but not yet populated by the fact that each cell in that row contains a single hyphen (-).

2. If you want to add a row *before* the selected cell, click the **Add a new grammar row before selection** icon () in the main toolbar.

   If you want to add a row *after* the selected cell, click the **Add a new grammar row after selection** icon () in the main toolbar.

3. Continue to add rows until you have enough for each entry you want to define.

To delete a row from the static grammar table:

1. Click a cell in the row you want to delete.
2. Click the **Delete selected grammar row** icon ().

Adding (or Deleting) Columns to a Static Grammar Table

By using columns, the way that an ASR server processed and interprets caller responses can be expanded and refined. Each column in a static grammar table represents a potential system response to a caller utterance.

Adding columns provide the following capabilities:

- Instruct the ASR server to ignore irrelevant input and listen only for valid responses.

  To do this, in the first row of a column where you want to ignore any but relevant responses, enter the key word **filler**.

  Note that you must enter this key word in the first row of the column. Once you enter the word **filler** in the first row, you cannot create entries in any other cell of that column. Dialog Designer displays **FILLER** in the top cell and clears the contents of any other cells in that column.

  Note:
  Exercise caution when using this option. When you use this key word, you are instructing the system, in effect, to expect irrelevant speech before a recognizable utterance. If the caller then does not, in fact, utter any irrelevant speech, the system can fail to recognize the valid utterance because it expects first to receive irrelevant speech.

- Collect multiple responses with a single grammar.

  Each new column represents another utterance to which the system can respond. So, if you have multiple columns of entries, the speech application then expects a caller response that matches an entry in each column. For this reason, exercise some caution when creating a grammar with multiple columns: Make sure that your callers will provide enough responses to satisfy the expectations of each expected input column.

  For example, if you have one column that lists five possible food items and another column that lists five possible drinks, the system then expects the caller to order both a food item and a drink. If the caller does not order both, the system treats it as a No Match situation and responds accordingly.

- Reuse the contents of one column to recognize a second input

  In cases where you want to offer the chance for a caller to provide two responses from the same list, you can create a column that refers to another column. For example, if you are offering a list of food items that a caller can order, you might want to list:

  – Hamburger
  – Cheeseburger
  – French fries
  – Onion rings
  – Salad

  If you offer this list in only one column, the caller can order only one item. If you want to offer the caller the chance to order two items at once, you can create a second column that refers to the first column.

To do this, in the first row of the column you want to use to refer to another column, enter the percent symbol (%) followed by the number of the column that you want to allow the chance to select from again. In our example, this means you would enter in the first row of the second column: **%1**

Note that you must enter the percent symbol and the number in the *first* row of the column. Once you enter these in the first row, you cannot create entries in any other cell in that column. Dialog Designer clears the contents of any other cells in that column.

In effect, this is a different way of collecting multiple responses from callers, so the cautions that apply to the description in the previous section also apply here.

To add a column to the static grammar table:

1. Click an active cell in the table.
2. If you want to add a column *before* the selected column, click the **Add a grammar column before selection** icon () in the main toolbar.

   If you want to add a column *after* the selected column, click the **Add a grammar column after selection** icon () in the main toolbar.

To delete a column from the static grammar table:

1. Click a cell in the column you want to delete.
2. Click the **Delete selected grammar column** icon ().

Using the TAG Option

The W3C Speech Recognition Grammar Specification (SRGS), version 1.0, recommends the use of tags with grammar entries. Although they are optional, the use of tags can greatly simplify the recognition of spoken responses, especially when you create multilingual applications.

For example, even when you want to solicit a simple “yes” or “no” response from callers, you should realize that caller responses might include “yeah,” “OK,” “yep,” and “sure” in place of “yes.” Similarly, negative responses might include “nah,” “nope,” or “uh-uh” in place of “no.”

You can use a single tag, “YES”, to assign the same value to all the possible affirmative responses. You can also use a single tag, “NO”, for all the possible negative responses.

Then, at run time, when the ASR server receives a spoken response, the server returns the tag as the **interpretation** and **value** for the input fields. This means that the application does not have to match up any of several different returns to determine the next step, but only a single one.

The use of tags can be particularly helpful when you want to create multilingual applications. This is true because you can accept second-language responses, tag them with the same tag as those in the first language and route the call flow appropriately. For example, in addition to all the English responses tagged as “YES” mentioned previously, you can also take the equivalent responses of “ja,” “ja wohl,” “klar,” “klar doch,” “todsicher,” and the like, and tag them to return a value of “YES”.

Tags facilitate your ability to use that return to further direct the caller, because tags make it easier to branch or determine the direction of the application. This is because you need only map one response, in this example “YES,” instead of having to map multiple responses, one for each alternate possibility.

To add an SRGS tag to a static grammar entry:

1. Double-click in the cell of the entry you want to tag.

   Dialog Designer highlights the entry and displays the tag assignment field on the right side of the cell.

2. Perform one of the following actions:

   - To create a new tag, highlight the default text **TAG** and enter the tag text you want.
   - Use the down arrow to select the tag you want to assign to the entry.

   Note:
   You can only perform this second action if you have already created one or more tags.

To continue the previous example: To tag all the affirmative responses as “yes,” assign a new tag, **yes**, to one entry in the table by entering **yes** in the tag field. For the remaining entries, select **yes** from the drop-down tag list.

Setting Static Entry Properties

For each entry in a static grammar, the following properties can be set in the **Avaya Properties** view (see the Speech Recognition Grammar Specification Version 1.0):

- **weight** - This property is a multiplying factor that indicates how likely it is that the entry will be uttered. A weight of 1.0 is considered neutral. That is, an entry with a weight of 1.0 does not bias the recognition one way or the other. A weight of greater than 1.0 biases the grammar entry in a positive way. A weight of less than 1.0 biases the entry in a negative way.

  The exact application of weights depends on the ASR engine being used. Therefore, a weight that works well on a particular platform might produce different results on other platforms.

  Weights have no effect on DTMF grammars.

  The default for this property is **-1**, which means that no weight is applied at all.

- **repeat** - This property indicates how many times you expect this entry might be repeated before anything in the next column.

  For example, suppose you have a grammar for which a caller can order a pizza. When ordering you expect that the caller might say “pizza,” “a large pizza,” or “a very large pizza.” Because you do not know whether or not the caller will say “large” or “very large,” you would set up column 1 with those options as possibilities. You would then assign a **repeat** value of **0-1** for both entries. The second column would contain a single entry, “pizza.”

  A repeat value of **0-** indicates that an item might be repeated zero times, once, or any number of multiple times. A repeat value of **1-** indicates that an item might be repeated once or any number of multiple times.

  The default for this property is **-1**, which means that this property is not applied at all.

- **repeat probability** - This property indicates how likely it is that this column will be part of the utterance. A value of 1.0 in this field indicates an overwhelming probability that the item will be uttered. A value of 0.0 in this field indicates an overwhelming probability that the item will not be uttered.

  For example, if you think there large chance that the item will be uttered, you might enter **0.75** in this field to alert the recognizer that it is likely to encounter this item.

  The valid range for this property is **0.0** through **1.0** or **-1**. The default for this property is **-1**, which means that this property is not applied at all.

Editing the Dynamic Grammar Java Class

When you create a dynamic grammar, Dialog Designer automatically creates a “skeleton” grammar Java class and opens the Java class editor. To complete the Java class that will generate the dynamic grammar, you must edit and save this grammar file.

When defining column names/slots, variables are specified from the grammar interpretation which are going to be returned to the application. If none are specified, then the grammar interpretation returns the standard interpretations (utterance, confidence, input mode, value, and interpretation). Use the **Move Up** and **Move Down** buttons to manage the grammar data in the list box, as necessary.

For more information about the skeleton Java class, see the Dialog Designer **Programmer Reference** documentation, **API Reference** > **DynamicGrammar** (under All Classes).

For general information about editing Java classes, see the Eclipse documentation.

Editing External Grammar Access Properties

Because an external grammar is not created and does not reside within the Dialog Designer environment, if an external grammar is used, you must only instruct the Dialog Designer speech application where to locate the grammar and what type of grammar it is.

The Grammar File Editor for an external grammar has the following fields:

External Grammar File Editor Properties

Field/Option	Description
URI	Name and port, in URL format, to the system that must gain access to the external grammar. This name and port information must be of the format: ***http://localhost:nnnn*** where: ● *localhost* is the IP address or fully qualified domain name of the computer on which the external grammar resides. ● *nnnn* is the port number the Dialog Designer application is to use to gain access to the grammar.
Media Type	An Internet mail extension (MIME) type. Select the media type that applies to the external grammar, or enter your own media type in the field. If you need to enter another media type, see your ASR server documentation for the appropriate type, and enter it in this field. **Note**: Currently, Dialog Designer has only one predefined media type: **applications/srgs+xml**. This type supports the SRGS specification.
Test	This option uses the information entered in the External Grammar Editor to try and reference the grammar from the designated application server and display it in a Web page (assuming the application server is running). This tests that you entered the path to the grammar correctly.
Column/Slots Names	When defining the column names/slots, variables are specified from the grammar interpretation which are going to be returned to the application. If none are specified, then the grammar interpretation returns the standard interpretations (utterance, confidence, input mode, value, and interpretation). Use the **Move Up** and **Move Down** buttons to manage the grammar data in the list box where slots are defined, as necessary.

Selecting Built-in Grammar Types

A built-in grammar is a static DTMF or voice (speech) grammar. Built-in grammar entries cannot be created or edited. Built-in grammar entries can only be selected for use.

Dialog Designer supports the following DTMF and Voice built-in grammars, as specified by the W3C VoiceXML 2.0 Recommendation. These types are selectable after a grammar file is initially created with a “built-in” style selected, within the Grammar File Editor.

Note:
Different Speech Servers may return different values for their supported built-in grammars. Refer to the documentation of your Speech Server on how they implemented built-in grammars.

Supported Built-In Grammar Types

Type	Supported Languages	Description
boolean	English Japanese Chinese	Inputs include affirmative and negative phrases appropriate to the current language. DTMF 1 is affirmative and 2 is negative. With this type, define what DTMF key presses signify a “yes” response and a “no” response, as follows in the Grammar File Editor: ● **Yes Digit** - Enter the DTMF key for a “yes” response. ● **No Digit** - Enter the DTMF key for a “no” response. If the field value is subsequently used in <say-as> with the interpret-as value “vxml:boolean”, it will be spoken as an affirmative or negative phrase appropriate to the current language.
number	English Japanese Chinese	Valid spoken inputs include phrases that specify numbers, such as “one hundred twenty-three”, or “five point three”. Valid DTMF input includes positive numbers entered using digits and “*” to represent a decimal point. The result is a string of digits from 0 to 9 and may optionally include a decimal point (“.”) using the star key (*****). **Note:** Do not allow callers to use a leading zero (0) with this option, as that might cause the interpreter to interpret this as an octal number. If the field is subsequently used in <say-as> with the interpret-as value “vxml:number”, it will be spoken as a number appropriate to the current language.

Supported Built-In Grammar Types (continued)

Type	Supported Languages	Description
digits	English Japanese Chinese	Valid spoken or DTMF inputs include one or more digits, 0 through 9. The result is a string of digits. With this type, a digit length for the expected DTMF key presses should be defined, as follows in the Grammar File Editor: • **Min Len** - Minimum number of DTMF key presses for a match. • **Max Len** - Maximum number of DTMF key presses for a match. • **Exact Length** - To restrict the number of key presses, complete this field. If the result is subsequently used in <say-as> with the interpret-as value “vxml:digits”, it will be spoken as a sequence of digits appropriate to the current language. A user can say for example “two one two seven”, but not “twenty one hundred and twenty-seven”.
currency	English	Valid spoken inputs include phrases that specify a currency amount. For DTMF input, the “*” key will act as the decimal point. The result is a string with the format UUUmm.nn, where UUU is the three character currency indicator according to ISO standard 4217, or mm.nn if the currency is not spoken by the user or if the currency cannot be reliably determined (for example, “dollar” and “peso” are ambiguous). If the field is subsequently used in <say-as> with the interpret-as value “vxml:currency”, it will be spoken as a currency amount appropriate to the current language.
date	English	Valid spoken inputs include phrases that specify a date, including a month day and year. DTMF inputs are four digits for the year, followed by two digits for the month, and two digits for the day. The result is a fixed-length date string with format yyyymmdd, such as “20000704”. If the year is not specified, yyyy is returned as “????”; if the month is not specified mm is returned as “??”; and if the day is not specified dd is returned as “??”. If the value is subsequently used in <say-as> with the interpret-as value “vxml:date”, it will be spoken as date phrase appropriate to the current language.

Supported Built-In Grammar Types (continued)

Type	Supported Languages	Description
phone	English	Valid spoken inputs include phrases that specify a phone number. DTMF asterisk “*” represents “x”, for extension. For example, a string containing a telephone number and optionally the character “x” to indicate a phone number with an extension. For example, in North America, the following entry 18005551234*103 tells the system to dial 1-800-555-1234, extension 103. If the field is subsequently used in <say-as> with the interpret-as value “vxml:phone”, it will be spoken as a phone number appropriate to the current language.
time	English	Valid spoken inputs include phrases that specify a time, including hours and minutes. The result is a five character string in the format hhmmx, where x is one of “a” for AM, “p” for PM, “h” to indicate a time specified using 24 hour clock, or “?” to indicate an ambiguous time. Input can be via DTMF. Because there is no DTMF convention for specifying AM/PM, in the case of DTMF input, the result will always end with “h” or “?”. If the field is subsequently used in <say-as> with the interpret-as value “vxml:time”, it will be spoken as a time appropriate to the current language.

Chapter 11: Language and Localization Considerations

Localization is the process of adapting a speech application project for use with other languages, to be used in different locales around the world.

To facilitate the localization of Avaya Dialog Designer speech projects and applications, Avaya designed Dialog Designer to separate language-related functions and components from the logic functions and components. This fact means you can create a single application, using a single call flow and business logic, but to easily and quickly adapt it for use with other languages.

This chapter describes all aspects of language and localization administration in Dialog Designer. It contains the following sections:

Language Implementation in Dialog Designer

Dialog Designer supports language implementation on four levels:

- **Project Languages** – The project language refers to the default starting language assigned when a speech project is created. In Dialog Designer you can create and add project languages, which are primarily a convenient way to assign the ASR language, the TTS language, and the localization bundle language all at one time. Adding project languages also copies all existing prompts and phrases to a new language directory, so that you can translate them to another language.

  See Administering Project Languages.

- **Automated Speech Recognition (ASR) Languages** – ASR languages are not installed as part of the Dialog Designer software. The ASR languages available to you depend on what languages are installed and available on your ASR server. When you add an ASR language to Dialog Designer, you must ensure that the language exists on the ASR server, or you might get unpredictable results with ASR requests.

  See Administering Automated Speech Recognition (ASR) Languages.

- **Text-to-Speech (TTS) Languages** – TTS languages are not installed as part of the Dialog Designer software. The TTS languages available to you depend on what languages are installed and available on your TTS server. When you add a TTS language to Dialog Designer, you must ensure that the language exists on the TTS server, or you might get unpredictable results with TTS requests.

  See Administering Text-to-Speech (TTS) Languages.

- **Localization Bundle Languages** – Language localization bundles are packaged with Dialog Designer, but not installed automatically during initial installation and configuration. If you find you need a localization bundle for your speech project, you can install it.

  The current release of Dialog Designer includes the *U.S. English* language localization bundle.

  Additional localization bundles can be obtained and installed as they become available. To receive updated information about the availability of localization bundles, contact your Avaya service representative or visit support.avaya.com.

  See Administering Localization Bundles.

Administering Project Languages

By default, each Dialog Designer project has one default language, until you add another project language. The default language is whatever you define as the program language when you create the project using the Speech Project wizard.

The Dialog Designer default is **english**. This language is the starting language that Dialog Designer loads at run time for the application to use.

When adding a project language, Dialog Designer creates a new language directory in the project. At the same time, Dialog Designer copies any existing grammar, phrase, and prompt files from the language directory on which you are basing the new package into the new directory. This makes it easier to identify and translate all the grammars, phrases, and prompts for the new language.

Tip:
If you know you are going to need an application to run in two different languages, it is a good idea to wait until the first language version of the project is done before adding a second project language. This ensures that all the required grammars, phrases, and prompts are copied into the new language directory.

Important:
In order for the call flow to use the prompt and grammar resources, it is essential to remember that prompt and grammar files *must not* be moved or renamed. It is important to remember that each project language must have the same number of prompt and grammar resource files, and the files must have the same names.

For example, suppose you have a project that is based on U.S. English, but you want to localize it for Canadian French. The first step, after finishing the project application, is to add a project language for Canadian French. See Adding a Project Language.

After adding the project language, then, you must go through all the grammars, phrases, and prompts in the Canadian French language resource directory and translate them all from English to French. If you are using audio variables, you must also install the localization bundle for Canadian French. See Installing a Localization Bundle.

As the last step, you must instruct Dialog Designer to use the Canadian French language resources, instead of the English resources. See Changing the Project Default Language.

The following sections describe the language administration options:

- Adding a Project Language
- Editing or Uninstalling a Project Language
- Deleting a Project Language
- Changing the Project Default Language
- Changing the Default Language within an Application

Adding a Project Language

To add a project language:

1. In the **Navigator** view, right-click the project directory. From the pop-up context menu, click **Properties**.

   Dialog Designer displays the **Properties for *projectName*** dialog box, where *projectName* is the name of your project.

2. In the contents pane on the left, select **Dialog Designer**.
3. Select the **Languages** tab.

4. In the **Project Languages** area, click **Add**. Dialog Designer displays the **Project Language** dialog box.
5. Complete the fields in the **Project Language** dialog box, as described in the following table.

   Note:
   An ASR language, TTS language, or Localization language must be added, of course, before it appears in the respective drop-down lists.

   Note:
   Installing standard phrases involves copying phrasesets and optionally the audio files. If the audio files will be stored externally, then it is up to the developer to store the audio files on the external server and configure the phraseset to reference the external location.
6. When done, click **OK**.

Adding a Project Language Dialog Box

Field	Description
Language Name	Name to identify the project language in Dialog Designer. This name only appears in Dialog Designer.
Base Language	The project language to use as the base for this project language.
ASR Language	The ASR language to use with this project language. Normally, this language is the same as the default project language.
TTS Language	The TTS language to use with this project language. Normally, this language is the same as the default project language.
Localization Language	The localization bundle to use with this project language. Normally, this language is the same as the default project language.
Standard Phrases	To install standard phrase audio files with this project language, click **Install**. Once installed, the newly installed language, with the project language name assigned in **Language Name**, is displayed within the **Project Languages** table.

Editing or Uninstalling a Project Language

To edit or uninstall a project language:

Note:
You cannot edit the name of the project language (**Language Name**) or the default language for the project language.

1. In the **Navigator** view, right-click the project directory. From the pop-up context menu, click **Properties**.

   Dialog Designer displays the **Properties for *projectName*** dialog box, where *projectName* is the name of your project.

2. In the contents pane on the left, select **Dialog Designer**.
3. Select the **Languages** tab.
4. In the **Project Languages** list, select the project language you want to edit.
5. Click **Edit**. Dialog Designer displays the **Project Language** dialog box.
6. Edit the fields in the **Project Language** dialog box, as described in the following table.

Note:
An ASR language, TTS language, or Localization language must be added, of course, before it appears in the respective drop-down lists.

Note:
Installing standard phrases involves copying phrasesets and optionally the audio files. If the audio files will be stored externally, then it is up to the developer to store the audio files on the external server and configure the phraseset to reference the external location.

7. When done, click **OK**.

Editing a Project Language Dialog Box

Field	Description
Language Name	Name of the project language, as listed in the **Project Language** table.
ASR Language	The ASR language to use with this project language. Normally, this language is the same as the default project language.
TTS Language	The TTS language to use with this project language. Normally, this language is the same as the default project language.
Localization Language	The localization bundle to use with this project language. Normally, this language is the same as the default project language.
Standard phrases	To install (or uninstall) the standard phrase audio files with this project language, click **Install** (or **Uninstall**).

Deleting a Project Language

To delete a project language:

1. In the **Navigator** view, right-click the project directory. From the pop-up context menu, click **Properties**.

   Dialog Designer displays the **Properties for *projectName*** dialog box, where *projectName* is the name of your project.

2. In the contents pane on the left, select **Dialog Designer**.
3. Select the **Languages** tab.
4. In the **Project Languages** list, select the project language you want to delete.
5. Click **Delete**.

   Dialog Designer deletes the selected project language, including the language directory and all associated language resources.

Changing the Project Default Language

The following procedure changes the default, or starting, language for a project. If you want to change languages within an application, in effect making it a multilingual application, you must write some Java code and employ that within your application. See Changing the Default Language within an Application.

To change the default language for a speech project:

1. In the **Navigator** view, right-click the project directory. From the pop-up context menu, click **Properties**.

   Dialog Designer displays the **Properties for *projectName*** dialog box, where *projectName* is the name of your project.

2. In the contents pane on the left, select **Dialog Designer**.
3. Select the **Web Descriptor** tab.
4. Select the entry that has **sage.startlanguage** as the entry in the **Name** column.
5. Click **Edit**.

   Dialog Designer displays the **Edit context parameter** dialog box.

6. In the **Value** field, enter the name of the language to use as the new default language for the application.

CAUTION:
Make sure the language name you enter in this field is a valid project language for the project. Otherwise, if the language does not exist or is named differently, the application generates run-time errors and the application does not execute properly.

7. To exit the **Edit context parameter** dialog box, click **OK**.
8. In the **Properties for *projectName*** dialog box, click **OK**.

Changing the Default Language within an Application

You can use the following procedure to change the default application language within the application. With this procedure, in effect, you can create multilingual applications.

To change the default language within an application:

1. Place a node in the main call flow, at the point where you want to change languages.
2. Right-click the node you just placed.
3. From the pop-up context menu, select **Edit *NodeName*.java**, where *NodeName* is the name assigned to the node. Dialog Designer displays the Java code editor for the node.
4. Place your cursor at the beginning of the Java code, just after the first line, which says: **package flow;**
5. Enter the following code on the line where you placed the cursor:

   import com.avaya.sce.runtimecommon.SCESession;

   Tip:
   Just to keep things tidy and make it easier to read the code, it is a good idea to make sure there is a blank line both before and after this line of code.
6. Scroll down and locate the lines of code that read as follows:

   /*

   *** Returns the Next item which will forward application execution**

   *** to the next form in the call flow.**
7. Place your cursor on the blank line just above these lines of code.

8. Enter the following code:

   public void requestBegin(SCESession session) {

   session.setCurrentLanguage("*language*");

   }

   where *language* is the name of the language you want to change to. This must be entered exactly as it appears in the project properties.

9. To save your changes, with the focus in the Java code editor, press **Control+S**.
10. Close the Java code editor.

Administering Automated Speech Recognition (ASR) Languages

Automated speech recognition (ASR) languages are not installed as part of the Dialog Designer software. Dialog Designer applications, however, generate the code to use ASR on the designated ASR server. This means that you must know what languages are installed and available on your ASR server.

By default, Dialog Designer projects use U.S. English (**en-us**) as the ASR language. You can, however, change the Dialog Designer project to use another ASR language.

The following sections describe the ASR language administration options:

- Adding an ASR Language
- Editing a Project Language to Use a Different ASR Language
- Deleting an ASR Language

Adding an ASR Language

When you add an ASR language to Dialog Designer, you must ensure that the language exists on the ASR server, or you might get unpredictable results with ASR requests.

To add an ASR language:

1. From the **Window** menu, select **Preferences**.

   Dialog Designer displays the **Preferences** dialog box.

2. In the navigation (left) pane of the **Preferences** dialog box, expand the **Avaya Dialog Designer** entry, and then select **Languages**.

   The right pane of the **Preferences** dialog box displays the **Languages** options.

3. In the **Automated Speech Recognition** box, click **Add**.

   Dialog Designer displays the **Add Automated Speech Recognition (ASR) Language** dialog box.

4. In the field, enter the code for the ASR language to add.

   This code must be of the format ***ll-cc***, where *ll* represents a two-character language code, and *cc* represents a two-character country code. For example, the code for U.S. English is **en-us**, and the code for UK English is **en-uk**.

   Note:
   This language code must match exactly the ASR language code of a language that you know is installed on the ASR server, or the system cannot process ASR requests that use this language.

   For a complete list of the two-letter country codes supported by the International Organization for Standardization (ISO 3166), see the ISO Web site countries list.

   For a complete list of the two-letter language codes supported by the ISO (ISO 639), see the ISO Web site languages list.

5. Click **OK**. Dialog Designer adds the new ASR language to the list of ASR languages.
6. In the **Preferences** dialog box, click **OK**.

Editing a Project Language to Use a Different ASR Language

To edit a project language to use a different ASR language:

1. In the **Navigator** view, right-click the project directory. From the pop-up context menu, click **Properties**.

   Dialog Designer displays the **Properties for *projectName*** dialog box, where *projectName* is the name of your project.

2. In the contents pane on the left, select **Dialog Designer**.

   In the display area on the right, Dialog Designer displays the **Dialog Designer** properties tabs.

3. Select the **Languages** tab.
4. In the **Project Languages** list, select the project language in which you want to change the ASR language setting.
5. Click **Edit**.

   Dialog Designer displays the **Project Language** dialog box.

6. From the **ASR Language** drop-down list, select the ASR language to change to.
7. In the **Project Language** dialog box, click **OK**.
8. In the **Properties for *projectName*** dialog box, click **OK**.

Deleting an ASR Language

To delete an ASR language:

1. From the **Window** menu, select **Preferences**.

   Dialog Designer displays the **Preferences** dialog box.

2. In the navigation (left) pane of the **Preferences** dialog box, expand the **Avaya Dialog Designer** entry, and then select **Languages**.

   The right pane of the **Preferences** dialog box displays the **Languages** options.

3. In the **Automated Speech Recognition** box, select the language you want to delete.
4. Click **Delete**. Dialog Designer displays a **Question** dialog that asks you to confirm the deletion.
5. To confirm the deletion, click **OK**. Dialog Designer displays a **Warning** dialog that informs you that removing the language might invalidate projects.
6. To confirm the deletion and remove the ASR language, click **Yes**.
7. In the **Preferences** dialog box, click **OK**.

Administering Text-to-Speech (TTS) Languages

Text-to-Speech (TTS) languages are not installed as part of the Dialog Designer software. Dialog Designer applications, however, generate the code to use TTS on the designated TTS server. This means that you must know what languages are installed and available on your TTS server.

By default, Dialog Designer projects use U.S. English (**en-us**) as the TTS language. You can, however, change the Dialog Designer project to use another TTS language.

The following sections describe the TTS language administration options:

- Adding a TTS Language
- Editing a Project Language to Use a Different TTS Language
- Deleting a TTS Language

Adding a TTS Language

To add a TTS language if it does not already exist:

1. From the **Window** menu, select **Preferences**.

   Dialog Designer displays the **Preferences** dialog box.

2. In the navigation (left) pane of the **Preferences** dialog box, expand the **Avaya Dialog Designer** entry, and then select **Languages**.

   The right pane of the **Preferences** dialog box displays the **Languages** options.

3. In the **Text-to-Speech** box, click **Add**.

   Dialog Designer displays the **Add Text-To-Speech (TTS) Language** dialog box.

4. In the field, enter the code for the TTS language to add.

   This code must be of the format ***ll-cc***, where *ll* represents a two-character language code, and *cc* represents a two-character country code. For example, the code for U.S. English is **en-us**, and the code for UK English is **en-uk**.

   Note:
   This language code must match exactly the TTS language code of a language installed on the TTS server, or the system cannot process TTS requests that use this language.

   For a complete list of the two-letter country codes supported by the International Organization for Standardization (ISO 3166), see the ISO Web site countries list.

   For a complete list of the two-letter language codes supported by the ISO (ISO 639), see the ISO Web site languages list.

5. Click **OK**. Dialog Designer adds the new TTS language to the list of TTS languages.

6. In the **Preferences** dialog box, click **OK**.

Editing a Project Language to Use a Different TTS Language

To edit a project language to use a different TTS language:

1. In the **Navigator** view, right-click the project directory. From the pop-up context menu, click **Properties**. Dialog Designer displays the **Properties for *projectName*** dialog box, where *projectName* is the name of your project.

2. In the contents pane on the left, select **Dialog Designer**.

   In the display area on the right, Dialog Designer displays the **Dialog Designer** properties tabs.

3. Select the **Languages** tab.

4. In the **Project Languages** list, select the project language in which you want to change the TTS language setting.
5. Click **Edit**. Dialog Designer displays the **Project Language** dialog box.
6. From the **TTS Language** drop-down list, select the TTS language to change to.
7. In the **Project Language** dialog box, click **OK**.
8. In the **Properties for *projectName*** dialog box, click **OK**.

Deleting a TTS Language

To delete a TTS language:

1. From the **Window** menu, select **Preferences**. Dialog Designer displays the **Preferences** dialog box.
2. In the navigation (left) pane of the **Preferences** dialog box, expand the **Avaya Dialog Designer** entry, and then select **Languages**.

   The right pane of the **Preferences** dialog box displays the **Languages** options.
3. In the **Text To Speech** box, select the language you want to delete.
4. Click **Delete**. Dialog Designer displays a **Question** dialog that asks you to confirm the deletion.
5. To confirm the deletion, click **OK**. Dialog Designer displays a **Warning** dialog that informs you that removing the language might invalidate projects.
6. To confirm the deletion and remove the TTS language, click **Yes**.
7. In the **Preferences** dialog box, click **OK**.

Administering Localization Bundles

Localization bundles need to be installed only when you plan to use audio variables in your speech project, because the localization bundle is what tells Dialog Designer how to handle the data input to provide the correct audio output. See Audio Variable.

Localization bundles are sets of files for a particular language that include:

- A JAR (Java ARchive) file that contains the logic to tell Dialog Designer how to perform the conversion from variable data to audio output for that language
- A metadata file that the Prompt File Editor uses to display the formats that the localization bundle supports

- The standard phraseset (***.phraseset**) file and standard phrase audio (***.wav**) files associated with that language

These bundles are delivered in the form of JAR files. Those bundles that were available at the time of the release of Dialog Designer are included on your software CD. Additional localization bundles, as they become available, can be downloaded from the Avaya customer support site, support.avaya.com. If you are looking for localization languages not available with this release, Avaya recommends that you let your Avaya service representative know, and check the customer support site frequently.

Tip:
When you receive a new localization bundle, save it to a directory on your hard drive, and make note of where you saved it. You will need to know this later to add it to the Dialog Designer workbench environment.

The following sections describe the localization bundle administration options:

- Adding a Localization Bundle
- Deleting a Localization Bundle
- Installing a Localization Bundle
- Uninstalling a Localization Bundle

Adding a Localization Bundle

Before you can install a language localization bundle, you must add it to Dialog Designer, if it has not already been added.

To add a localization bundle:

1. From the **Window** menu, select **Preferences**.

   Dialog Designer displays the **Preferences** dialog box.

2. In the navigation (left) pane of the **Preferences** dialog box, select **Dialog Designer > Speech > Languages**.

3. In the **Audio Localization Packages** box, click **Add**.

   Dialog Designer displays the **Browse for localization package** dialog box.

4. Locate the localization bundle JAR file you want to add and click **Open**.

   Dialog Designer automatically adds the localization bundle to the workbench environment. The new localization bundle is displayed in the **Audio Localization Packages** list.

5. In the **Preferences** dialog box, click **OK**.

Deleting a Localization Bundle

CAUTION:
When you delete a localization bundle, Dialog Designer removes the bundle completely from the development environment. Deleting localization bundles can also invalidate languages existing in the application, because the file that the Prompt File Editor uses for audio variable formats no longer exists. Finally, this procedure can result in code generation errors, because the localization language code no longer exists. Avaya recommends that you exercise extreme caution when performing this procedure.

To delete a localization bundle:

1. From the **Window** menu, select **Preferences**. Dialog Designer displays the **Preferences** dialog box.
2. In the navigation (left) pane of the **Preferences** dialog box, select **Dialog Designer > Speech > Languages**.
3. In the **Audio Localization Packages** box, select the localization bundle you want to delete.
4. Click **Delete**. Dialog Designer displays a **Question** dialog that asks you to confirm the deletion.
5. To confirm the deletion, click **OK**. Dialog Designer displays a **Warning** dialog that informs you that removing the localization bundle might invalidate projects.
6. To confirm the deletion and remove the localization bundle, click **Yes**.
7. In the **Preferences** dialog box, click **OK**.

Installing a Localization Bundle

Before installing a language localization bundle, it must be added to Dialog Designer, if it has not already been added. To add a localization bundle, see Adding a Localization Bundle.

Note:
The following procedure installs the Java code that contains the logic for language localization. This procedure does not install the standard phrases for the localization language. That is a separate procedure. Both standard phrases and localization logic must be installed for Audio Variables to work in a project. See Using a Localization Bundle's Standard Phrasesets.

To install a localization bundle:

1. In the **Navigator** view, right-click the project directory. From the pop-up context menu, click **Properties**.

   Dialog Designer displays the **Properties for *projectName*** dialog box, where *projectName* is the name of your project.

2. In the contents pane on the left, select **Dialog Designer**.

   In the display area on the right, Dialog Designer displays the **Dialog Designer** properties tabs.

3. Select the **Languages** tab.
4. In the **Localization Bundles** display list, select the localization bundle to install.
5. Click **Install**. The **Installed** column status for the localization bundle changes to **installed**.
6. Click **OK**.

Uninstalling a Localization Bundle

Note:
Unlike the procedure to delete a localization bundle, which completely removes the bundle from the development environment, the following procedure removes only the Java code library that contains the logic for language localization. Thus, the impact of performing the following procedure is not as great. Note also that this procedure does not uninstall the standard phrases.

To uninstall a localization bundle:

1. In the **Navigator** view, right-click the project directory. From the pop-up context menu, click **Properties**.

   Dialog Designer displays the **Properties for *projectName*** dialog box, where *projectName* is the name of your project.

2. In the contents pane on the left, select **Dialog Designer**. In the display area on the right, Dialog Designer displays the **Dialog Designer** properties tabs.
3. Select the **Languages** tab.
4. In the **Localization Bundles** display list, select the localization bundle to uninstall.

   Note:
   Verify that the **Installed** column shows a status of **installed** for the selected localization bundle. If the status is **uninstalled**, the **Uninstall** button is disabled.

5. Click **Uninstall**. The **Installed** column status for the localization bundle changes to **uninstalled**.
6. Click **OK**.

Using a Localization Bundle's Standard Phrasesets

Standard phrasesets are used, in conjunction with language localization bundles, to provide audio variable functionality to Dialog Designer applications. In audio variables, Dialog Designer takes the value of a variable and parses it in such a way that pre-recorded audio files can be used to play back certain standardized types of information. For more information about audio variables, see Audio Variable.

Note:
Installing standard phrases involves copying phrasesets and optionally the audio files. If the audio files will be stored externally, then it is up to the developer to store the audio files on the external server and configure the phraseset to reference the external location.

To install a set of standard phrasesets:

1. In the **Navigator** view, right-click the project directory. From the pop-up context menu, click **Properties**.

   Dialog Designer displays the **Properties for *projectName*** dialog box, where *projectName* is the name of your project.

2. In the contents pane on the left, select **Dialog Designer**.

   In the display area on the right, Dialog Designer displays the **Dialog Designer** properties tabs.

3. Select the **Languages** tab.

4. In the **Project Languages** list, select the language for which you want to install the standard phrases.

5. Click **Edit**. Dialog Designer displays the **Project Language** dialog box.

6. Click the cascading icon adjacent to Localization Bundle Options to display standard phrases installed (or not installed). From here, you can:

   - *Install standard phrases* - Click **Install**, and the indicated localization language is installed. To uninstall, subsequently click **Uninstall**.
   - *Indicate for standard phrases on an external server to be installed* - If this option is selected, it means that the audio files for the localization bundle will not be installed in the project.

     In this case, the phraseset file for the standard phrases must be edited to specify that the localization bundle uses "External Audio" files and the "Base URL" is set for the audio file location. See Adding a Localization Bundle for more details.

7. Click **OK** when done.

   Dialog Designer copies the files based on the specified localization bundle options to the appropriate resource directories.

Chapter 12: Working with Database Operations

Within Dialog Designer, databases can be used in call flows. For example, during a call, a caller might want to know the current status of an account, status which is maintained within an SQL database. A call flow can be written to query the database for the account information of the caller, including current status. Then this information can be assigned to audio or text variables to play the information back to the caller.

Dialog Designer uses database operation files to perform these database operations. To use database operation files, a database operation file must first be created and defined. Before creating a database operation file, however, one or more data sources in which to perform the operations must be configured. After the database operation file has been created and defined, it can be used within a Data node in a call flow application.

This chapter describes all aspects of creating and using database operation files. It contains the following sections:

Configuring Data Sources

Before performing any operations involving a database, Dialog Designer must be configured to access the database. Data sources are configured in Dialog Designer using the **Datasource** tab of the **Properties for *ProjectName*** dialog box.

Because Dialog Designer is a Java-based tool, data sources for Dialog Designer speech applications must be JDBC-compliant. If JDBC drivers are not installed on your computer, they must be installed to work with data sources. To install these drivers, see the documentation that cam with your database software.

For more information about configuring Dialog Designer to work with a JDBC database driver, see Database Preferences.

For more information about configuring data sources, see Database Tab.

When at least one data source is configured for the project, database operations can be created to make use of the data source in the project. To create database operations, use the Database Operation wizard. See Using the Database Operation Wizard to Create a Database Operation File.

Using the Database Operation Wizard to Create a Database Operation File

To create a database operation file, the Database Operation Wizard is used. Database operation files can be used to:

- Query a database and return the query results to one or more variables.
- Add, update, or delete records in a database.
- Run a procedure stored within a database.

To create a database operation file, see the following sections:

- Accessing the Database Operation Wizard
- Create a Database Operation Page
- Select Data Objects Page
- Map to Variables Page

Note:
If a database operation is attempted to be created before data sources are configured, Dialog Designer displays an error message. To configure data sources, see Database Tab.

After a database operation file is created, to edit database operation file configurations, see Using the Database Operation File Editor.

Accessing the Database Operation Wizard

To access the Database Operation Wizard, use one of the following methodologies:

- From the **File** menu select **New** > **Database Operation File**.
- On the main toolbar, click the icon for the Database Operation wizard ().
- Right-click anywhere in the Navigator view, then select **New** > **Database Operation File**.

Create a Database Operation Page

Configure the following settings in the Create a Database Operation page of the Database Operation Wizard.

Database Operation Wizard—Create a Database Operation Page Settings

Setting	Description
Available Projects	Select the project to which the database operation file will be applied.
Data Source name	Select the database (data source) to use for this operation.
Operation	Select the type of operation you want to perform. Options include: • **Add** - Takes the values of selected variables in Dialog Designer and creates a record in the database, populating the columns with the variable values. • **Delete** - Deletes designated records from the database. • **Execute** - Runs a procedure previously stored in the database. • **Query** - Retrieves the values of selected records in a database and assigns the values to variables in Dialog Designer. • **Update** - Updates the values of selected records in the database with new values from selected variables in Dialog Designer.
File name	Name identifying the database operation that describes the function of the database operation. For example, an operation to query a database for the account status for a caller might be called the **QueryAcctStatus**. **Note:** Database operation file names must follow Java naming conventions (see Naming Java Components).
Open file for editing	To automatically open the Database Operation File Editor when done creating the file within the Wizard, select this check box. When done creating the database operation file with the Database Operation wizard, the database operation is not yet finished. You must use the Database Operation File Editor to further define and set parameters for the database operation.

When done, perform one of the following actions:

- To select data objects for the database operation file, click **Next**. Dialog Designer displays the Select Data Objects Page.
- To exit the Database Operation Wizard without creating the database operation file, click **Cancel**.

Select Data Objects Page

Select the data objects to be included in the database operation file in the Select Data Objects page of the Database Operation Wizard. Check all data objects that are to be included with the database operation file.

When done, perform one of the following actions:

- To map selected columns or parameters to variables for the database operation file, click **Next**. Dialog Designer displays the Map to Variables Page.
- To exit the Database Operation Wizard without creating the database operation file, click **Cancel**.

Note:
If **Delete** is selected as the **Operation** type in Create a Database Operation Page, the Select Data Objects page is not displayed.

Map to Variables Page

Use the Map to Variables page of the Database Operation Wizard to:

- Select which columns or parameters to map to variables.
- Map columns in the data source to variables. When the **Operation** type is **Add** or **Update**, this mapping determines which variable values will be written to the database.
- Map parameters in the data source to variables. When the **Operation** type is **Query**, this mapping determines which records are retrieved from the database, the values of which the query operation stores in the mapped variables.

If Dialog Designer should not automatically create the variables to map to, custom variables must be created to map to *before* attempting to map any variables in the Database Operation Wizard. For information about creating custom variables, see Creating Variables.

The following table describes the options available in the Map to Variables page.

Database Operation File Wizard—Map to Variables Page Settings

Field	Description	Operation
Column Name	Select the columns that will perform the operation.	Add Query Update

Database Operation File Wizard—Map to Variables Page Settings (continued)

Field	Description	Operation
Column Type	Displays the database type for each column. This type helps determine what type of variable to use. The types for columns are defined by the database vendor. For more information about the types, see your database documentation.	Add Query Update
Parameter Name	Displays the parameter names for inputs that the stored procedure expects to receive.	Execute
Parameter Type	Displays the type of the expected input parameter.	Execute
Variable Name	Select the variable to map the column or parameter to. If the variable is a complex variable, select a **Variable Field**. If a variable is selected, the **Auto Create** option is not available.	Add Query Update Execute
Variable Field	If the variable selected in the **Variable Name** field is a complex variable, select a field to map the column or parameter.	Add Query Update Execute
Auto Create	Check this option to indicate that Dialog Designer should automatically create and name a variable to map the column or parameter to. If selected, the **Variable Name** and **Variable Field** fields are disabled.	Add Query Update Execute
Creating one complex variable for all the columns		
Result set returned	Also in case of creating the stored procedure operation that returns a resultset, you need to check the **Result set returned** option and specify the # of columns expected to return. Then when you click Next, another Map to Variables page would come up and allow you to map the resultset column to variables.	

When done making selections and settings on this page, click **Finish**.

Dialog Designer creates the database operation file and places it in the following directory: ***<ProjectName>*\connectivity\dboperations**. A suffix of **.dbop** is assigned to the new database operation file.

Also, if the **Open file for editing** check box on the first page of the wizard was selected, the Database Operation File Editor is opened. See Using the Database Operation File Editor.

Using the Database Operation File Editor

The Database Operation File Editor is used to further define database operations. In general, this is done by setting conditions in which the database operation file compares a constant or variable value in Dialog Designer with a specified value or set of values in a column of the database. The operation then operates only on the records that meet the specified conditions.

The Database Operation File Editor opens in the main editor workspace of Dialog Designer. For the procedures to open the Database Operation File Editor, see Accessing the Database Operation File Editor. For general information about page tabs, see Editor View Tabs.

The Database Operation File Editor can have two or three tabs associated with it, depending on the type of operation file that was created to perform. Furthermore, the fields and options for each tab vary somewhat, depending on what type of operation the file was created to perform. For more detailed information about these tabs, see:

- Accessing the Database Operation File Editor
- Database Operation File Editor—Database Operation Tab
- Database Operation File Editor—Predicate Tab
- Database Operation File Editor—SQL Query Tab

Accessing the Database Operation File Editor

To access the Database Operation File Editor, use one of the following methodologies:

- Create a database operation file with the Database Operation Wizard.

  If the **Open file for editing** check box is selected on the first page of the Database Operation wizard, Dialog Designer opens the Database Operation File Editor automatically upon clicking **Finish** to create the database operation file.

  To create a database operation file with the Database Operation wizard, see Using the Database Operation Wizard to Create a Database Operation File.

- In the Navigator view, double-click the ***.dbop** file you want to edit.
- For any database operation file that is already open in the Database Operation File Editor workspace but not currently displayed in the active workspace, click the Database Operation File Editor tab for that database operation file.

Database Operation File Editor—Database Operation Tab

For a query or insert operation on a table, you may need to adjust the columns that are affected by the operation without using the New Database Operation wizard to re-create the operation. For the **Select** operation, you can add another table to do a join, or to just add other columns from the same table. For the **Insert** operation, it makes sense to only add other columns from the same table as defined in the operation.

Use the following buttons below the table as follows:

- **Add** button – Upon clicking, a dialog window comes up with a list of data objects that you can select from, which comes from the same data source defined in the operation.
- **Remove** button – Upon clicking, whatever column selected in the table display is deleted.

Use the **Database Operation** tab of the Database Operation File Editor to:

- View basic information about the database operation file. See Viewing Basic Information.
- Remap variables to columns. See Remapping Variables to Columns.
- Determine the order in which return values are to be returned and sorted (**Query** operations only). See Determining Order for Return Results.

Viewing Basic Information

The Database Operation tab displays the following information about the database operation file:

Note:
Not all of the following information is displayed for all operation types. A **Delete** operation displays only the first four types. Other exceptions are as noted in the field descriptions.

Database Operation Tab—Viewing Basic Information

Field or Button	Description
Datasource name	Displays the name of the data source as it appears in Dialog Designer.
Operation	Displays the type of operation this file performs. For more information about operation types, see Create a Database Operation Page.
Distinct select	(**Query** operations only) Return only one record for all the records that have the same data in the returned columns.
Table name	Displays the name of the table in which the database operation is to operate.
Type	Displays whether the database operation performs within a **table** or executes a **procedure**.

Database Operation Tab—Viewing Basic Information (continued)

Field or Button	Description
Column Name	(Not in **Execute** operations - Display only) Displays the name of the column in which the operation is to be performed.
Parameter Name	(**Execute** operations only- Display only) Displays the names of all input and output parameters used by the stored procedure.
Parameter Type	(**Execute** operations only- Display only) Displays the type associated with each input and output parameter used by the stored procedure.
Data Type	Displays what type of data is contained within the column or parameter. ***Note:*** This type helps determine what type of variable to use.
Variable Name	Displays the name of the variable currently mapped to this column or parameter. For information about remapping variables, see Remapping Variables to Columns.
Variable Field	Displays the name of the field associated with the variable if the variable selected in the **Variable Name** field is a complex variable.
Order By	(**Query** operations only) Displays order selections for the results to be returned. For more information about this field, see Determining Order for Return Results.
... button	To change a data source name, click on this button. If the data source does not match the database schema, there will be runtime errors.

Remapping Variables to Columns

To remap a variable: In the **Variable Name** field, select the new variable to map to the selected column or parameter.

If the new variable is a complex variable, you must also select a field in the **Variable Field** drop-down list.

Determining Order for Return Results

With **Query** operations, you can determine the order in which the return results are presented.

For example, suppose you have a database operation that returns a list of all customers who opened accounts during the past week. The query returns each customer's first name, last name, and account number. You want the results returned and sorted by last name as the first column. If there are two or more customers with the same last name, you want the results then returned and sorted by first name. You do not want the returns sorted by account number at all.

To accomplish this, you would enter the number **1** in the **Order By** field for the last name and the number **2** in the **Order By** field for the first name. You would leave the **Order By** field for the account number blank.

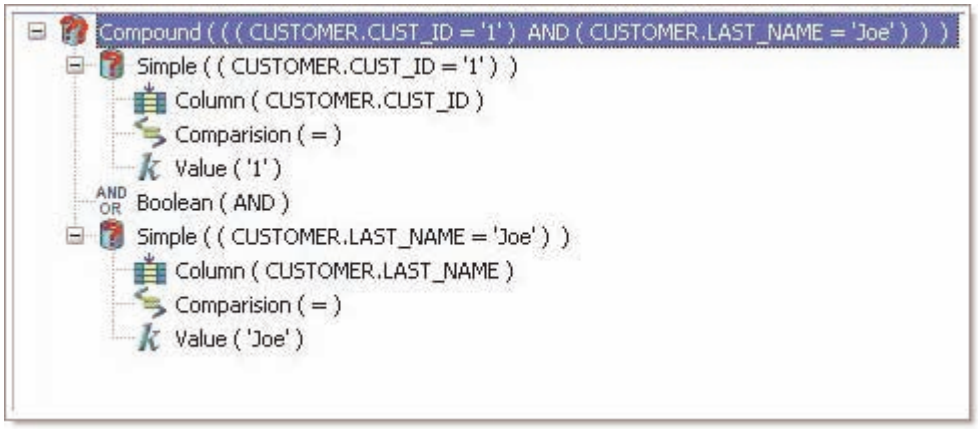
Database Operation File Editor—Predicate Tab

Use the **Predicate** tab to set the conditions that determine which records are operated on. For example, suppose you want the database operation to query the database and return all records for customers who are delinquent in their payments. You can use the **Predicate** tab to set the conditions and instruct the database which records to return.

Note:
The **Predicate** tab appears and is used only with **Delete**, **Query**, and **Update** operations.

Compound conditions can be created within the Predicate tab. These conditions are used for nesting two conditions (Simple, Join, or Compound) that are joined by a Boolean operator. In addition, compound conditions can be nested to build complex Where clauses. See the following example for better understanding.

Example of a Compound Condition



Setting Conditions for a Simple Database Operation

A **Simple** condition sets a single condition that affects the operation.

For example, suppose you want the database operation to return the names of customers who opened accounts within the past 30 days. You have already set up the database operation file to return the names of customers, but you want to use a **Simple** condition to return information for only those customers who have opened accounts in the past 30 days.

In a case like this, you would set up the condition to look at the column containing the date the account was opened, and return only those records for which the date is 30 days or less from the current date.

If you need to use two conditions for a single database operation, use the **Compound** condition setting. See Setting Conditions for a Compound Database Operation.

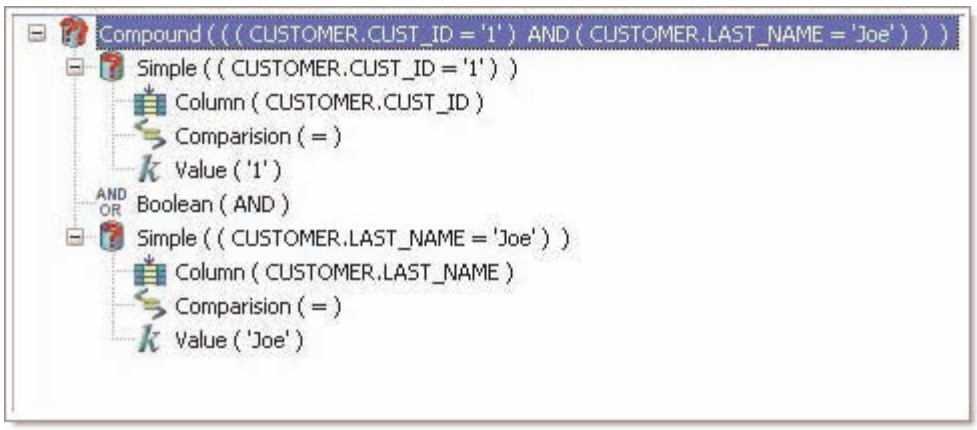
To set the conditions for a simple database operation:

1. From the palette, under **Condition**, select **Simple** and place it in the workspace.
2. From the palette, under **Operand**, select **Column** and place it in the workspace.
3. In the workspace, select the **Column** operand you just placed there, and then, in the **Avaya Properties** view, **Name** field, select the column to use in testing the condition.
4. From the palette, under **Operator**, select **Comparison** and place it in the workspace.
5. In the workspace, select the **Comparison** operator you just placed there, and then, in the **Avaya Properties** view, **Operator** field, select the operator to use in making the comparison between the column value and the variable or constant value.
6. From the palette, under **Operand**, select one of the following operands and place it in the workspace:
   - **Variable** - To compare the column value with a variable value, use this option.
   - **Value** - To compare the column value with a constant or absolute value, use this option.
7. In the **Avaya Properties** view, perform one of the following actions:
   - If you selected **Variable** as the second **Operand**, select from the **Name** drop-down list the variable to use for the comparison.

     If the variable is a complex variable, you must also select a field from the **Field** drop-down list.
   - If you selected **Value** as the second **Operand**, enter in the **Value** field the exact value to be used for the comparison.

Following is an example of a nested Simple condition.

Example of Nested Simple Conditions



Setting Conditions for a Compound Database Operation

A **Compound** condition sets two conditions that both affect the operation.

For example, suppose you want the database operation to return the names of customers who opened accounts within the past 30 days. You have already set up the database operation file to return the names of customers, but you want to use a **Compound** condition to return information for only those customers who have opened accounts in the past 30 days and who have an outstanding balance of more than $500.

In a case like this, you would set up a **Compound** condition with two **Simple** conditions: The first Simple condition looks at the column containing the date the account was opened, and return only those records for which the date is 30 days or less from the current date. The second Simple condition looks at the current balance column and returns only those customers from the first set of returns that have an outstanding balance of more than $500. The final returns from this operation would contain the names of only those customers who meet both conditions.

Note:
Each **Compound** condition permits you to set two and only two conditions. However, if you need to set more than two conditions, you can nest **Simple** and **Compound** conditions within other **Compound** conditions to attain the number and combination of conditions you need.

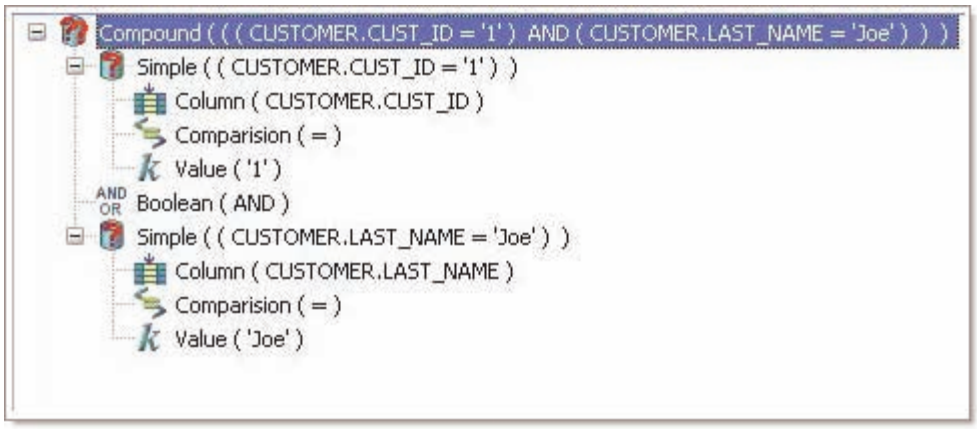
If you want to use only a single condition for a database operation, use the **Simple** condition setting. See Setting Conditions for a Simple Database Operation.

To set the conditions for a compound database operation:

1. From the palette, under **Condition**, select **Compound** and place it in the workspace.
2. Use the procedure in Setting Conditions for a Simple Database Operation to add two **Simple** conditions to the **Compound** condition.

Following is an example of a Compound condition.

Example of a Compound Condition



Database Operation File Editor—SQL Query Tab

The **SQL Query** tab shows the query code to be sent to the SQL server when this database operation file is executed. This is helpful to verify that the database operation does what you expect it to do. (This assumes that you are familiar with SQL query commands and syntax.) By testing the SQL query before actually running the application, the designer can find problems in the logic earlier in the design process.

The query can be tested by clicking the **Execute** button, with the query results being returned in the grid below the button. Notice that the columns in the result grid match the returned columns in the select statement.

By plugging in different values for variables (through simple or compound logic defined in the **Predicate** tab, the value(s) is displayed in the **Criteria Input** field on the **SQL Query** tab), and upon executing, you can test the query and see the results.

In general, if there is an error when executing a Database operation, a message box is displayed with the error message.

Note:
By right-clicking in the grid, a **Copy** option is available for copying selected rows into Notepad.

Examples of supported SQL database operations follow.

- Select SQL Query
- Insert SQL Query
- Delete SQL Query
- Update SQL Query.
- Stored Procedures SQL Query

With the combination of these supported operations, the database data can be easily manipulated without the need for an external tool:

Select SQL Query

For example:

```
SELECT Customer.Cust_ID, Customer.First_Name, Customer.Last_Name
  FROM dbo.Customer
  WHERE ( Customer.Cust_ID = $[CustomerSQL:Cust_ID} )
```

Criteria Input Example:

Cust_ID: **1**

Click **Execute** and the results grid indicates the First and Last Name for customer with the ID of 1 (*John* is the first name, and *Jacobs* is the last name in this example.)

Insert SQL Query

For example:

```
FROM dbo.Customer
  ( Cust_ID, First_Name, Last_Name )
  VALUES( $[AddCustomerSQL:Cust_ID], $[AddCustomerSQL:First_Name],
  $[AddCustomerSQL:Last_Name} )
```

Criteria Input Example:

Cust_ID: **3**

First_Name: **Jackie**

Last_Name: **Snow**

Click **Execute** and the results grid indicates the number of rows added (1, in this example).

Delete SQL Query

For example:

```
DELETE FROM dbo.Customer
  WHERE ( Customer.Cust_ID = ${CustomerSQL:Cust_ID} )
```

Criteria Input Example:

Cust_ID: **2**

Click **Execute** and the results grid indicates the number of rows deleted (1, in this example).

Update SQL Query.

For example:

```
UPDATE dbo.Customer
  SET First_Name = ${UpdateCustomerSQL:First_Name}
  WHERE ( Customer, Cust_ID = ${CustomerSQL:Cust_ID} )
```

Criteria Input Example:

First_Name: **Jackie**

Cust_ID: **3**

Click **Execute** and the results grid indicates the number of rows updated (1, in this example).

Stored Procedures SQL Query

Example 1:

```
{ ${SPSQL:_RETURN_VALUE} = call dbo.getCustomerById;
  1 ( ${SPSQL:_Param1) }
```

Criteria Input Example 1:

@Param1: **1**

Click **Execute** and the results grid indicates the result of the query.

Example 2:

```
{ call DDTEST,GETCUSTOMER( ${ExecuteOracleDB:PARAM1)
  ${ExecuteOracleDB:PARAM2} )
```

Criteria Input Example 2:

PARAM1: 2

Output Parameters:

Tom

Click **Execute** and the results grid indicates the result of the query.

Employing Database Operations in Call Flows

Once the database operation file has been created and defined, you can use it in your call flows. You can use database operation files only within a Data node.

To employ a database operation file in a call flow:

1. Place a Data node in the Call Flow Editor main workspace.
2. To open the Data node editor, double-click the Data node.
3. From the Data node editor palette, select the **Database** item and place it in the editor workspace.
4. In the **Avaya Properties** view, **Name** field, select the database operation file to use.

Chapter 13: Working with Web Services

With Avaya Dialog Designer, Web services can be employed within speech application 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. Finally, the Web service operation can be employed in a speech application call flow.

Note:
Dialog Designer supports Axis 1.4 SOAP to implement Java Web services.

This chapter describes all aspects of creating and using Web service operations in speech application call flows. It contains the following sections:

About 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.

Important:
When selecting a Web service, be careful to avoid selecting 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 tool *Tcpmon* for monitoring HTTP traffic. We recommend that this tool be used when debugging Web services. Documentation on how to use Tcpmon can be found at:
http://ws.apache.org/axis/java/user-guide.html#AppendixUsingTheAxisTCPMonitorTcpmon

Creating a Web Service Operation File

Web service operation files, used within a speech application project to access and use Web services, are created using the Web Service Operation wizard.

Access to the Web Service Operation wizard using one of the following methodologies:

- From the **File** menu select **New** > **Web Service Operation File**.
- On the main toolbar, click the icon for the Web Service Operation wizard ( ).
- Right-click anywhere in the Navigator view, then select **New** > **Web Service Operation File**.

Once displayed, create a new Web Service operation file by completing the Wizard fields described in the following table.

When done defining a Web Service using the Web Service Operation Wizard, Dialog Designer creates the database operation file and places it in the ***<ProjectName>*\connectivity\wsoperations** directory. Dialog Designer assigns the new database operation file a suffix of **.wsop**.

Creating a Web Service Operation File

Field/Option	Description
Specify General Parameters (Wizard Page 1) Specify Web service operation parameters.	
Available Projects	Select the project for which you want to create the Web service operation file.

Creating a Web Service Operation File (continued)

Field/Option	Description
File Name	Name of the Web Service being created.
Open file for editing	To automatically open the Web Service Operation File Editor when done creating the file with the wizard, select this check box.
WSDL URL	The URL of the WSDL file for the Web service in standard HTTP or HTTPS format. All Web services has an associated WSDL file that describes what services the Web service provides. Dialog Designer uses this WSDL information to populate the **Operation** field list. **Note:** Dialog Designer supports only WSDL files that contain unique method signatures. **Note:** Dialog Designer does not support operation overloading, or the use of methods with the same name but different parameters. **Tip:** If a Web service has been previously created, Dialog Designer remembers the URL. For future Web Service creations, Dialog Designer automatically populates this field with the URL. Dialog Designer keeps track of the last 20 Web services URLs used in operations.
Authentication	Select how Dialog Designer should authenticate with the Web Service host. Options are: • **No Authentication** • **Basic** – Transmits the username/password pair in an unencrypted form from browser to server in the HTTP header. • **Digest** – Transmits the username/password pair encrypted from the browser to server in the HTTP header.
User Name/Password	When Basic or Digest authentication is selected, these fields are displayed. Enter a username and optionally a password for authentication.
Load operations from WSDL file	Click this button to load the Web Services operations from the WSDL file.
Timeout	Sets the maximum time, in seconds, for fetching and parsing the WSDL file.
Use Document Wrapping	Multiple input and output parameters will be wrapped in a class if this option is checked.
Allow WSDL Imports	Allow the WSDL file to reference other files by using the WSDL import statement.

Creating a Web Service Operation File (continued)

Field/Option	Description
Operation	Select the service to be used in this operation. This assumes that the following information is known: • What services are offered by the Web service • What form the services take
Package for types	Specifies the package name for any Java beans that are generated as a result of defining a Web service operation. The default value is **connectivity.ws.beans**.
Mapping Input & Output Parameters (Wizard Page 2 & 3) Map input/output parameters expected by the Web service to the values of variables in Dialog Designer. For example, for a Web service that provides a weather report, the expected input might be the name of a city; the expected output would be the weather report string (mapped to the appropriate variable). **Note:** If you do not want Dialog Designer to automatically create the variables to map to the parameters, you must create custom variables *before* attempting to map any parameters. See Creating Variables. Otherwise, variables can be remapped later, using the Web Service Operation File Editor. See Viewing or Editing a Web Service Operation File.	
Parameter Name	Name of the input/output parameter defined in the Web service.
Parameter Type	Data type that the input/output parameter expects from Dialog Designer. Make sure that the variable assigned to each parameter is of the correct type.
Variable Name	The variable to map to the parameter. If a complex variable, also select a **Variable Field**. If a variable is selected in this field, the **Auto Create** option *cannot* also be selected.
Variable Field	Field to map to the parameter required if the variable selected in the **Variable Name** field is a complex variable.
Auto Create	To have Dialog Designer automatically create and name the variable mapped to the parameter, check this option. If selected, Dialog Designer clears the **Variable Name** and **Variable Field** values, if previously set.

Creating a Web Service Operation File (continued)

Field/Option	Description
Use Java Obj	Indicates whether the content of the variable should be treated as a Java object to prevent Dialog Designer from mapping it. Use this option when the structure of the data to be sent to the Web service is more complex than what can be contained within a normal variable. When using this option, subsequently, the following steps must be taken: ● Create the Java object and storing it in the variable, if sending the object to the Web service. ● Write the required Java code by overriding an existing method, such as the following method: **requestBegin (SCESession mySession)** ● Retrieve the Java object from the variable, if receiving the object from the Web service. See the following example section for more information. For more information about the Dialog Designer API and available methods, see the Dialog Designer *Programmer Reference* guide in the online Help system.

Example for When to Set the “Use Java Obj” Option

Use this option when the structure of the data to be sent to the Web service is more complex than what can be contained within a normal variable. For example, suppose a Web service requires an object with a structure like the following:

```
Person
   Name
      First
      Last
   Age
   Gender
```

Dialog Designer can handle only one level of nested variables using complex variables. That means, in this example, that Dialog Designer could send only the **Person**, **Name**, **Age**, and **Gender** values. Dialog Designer could not, however, return the second level, that is, the **First** and **Last** values.

By selecting this option when creating or editing a Web Service operation file, at run-time, the **Person** object is treated as a single Java object, including all nested values. You can use custom Java code then to send all the values for the object that are required.

When using this option, you are responsible for creating the Java object and storing it in the variable, when sending the object to the Web service. To do this, you must write the required Java code. Usually, this is done by overriding an existing method, such as the **requestBegin(SCESession mySession)** method. For more information about the Dialog Designer API and the methods you can use for this, see the Dialog Designer *Programmer Reference* guide in the online Help system.

Viewing or Editing a Web Service Operation File

The Web Service Operation File Editor is used to view or edit the Web Service Operation file. Using the file editor, you can:

- View information about the Web service operation file.
- Remap variables to parameters.
- Assign parameters to use Java objects.

Open the Web Service Operation File Editor using one of the following methodologies:

- Create a Web service operation file with the Web Service Operation wizard, and select the **Open file for editing** option. The Web Service Operation File Editor automatically opens upon finishing the Web service operation file creation.

  See Creating a Web Service Operation File.
- In the Navigator view, double-click the ***.wsop** file you want to edit.
- For any Web service operation file that is already open in the Web Service Operation File Editor workspace but not currently displayed in the active workspace, click the Web Service Operation File Editor tab for that Web service operation file.

Once open, the Web Service Operation File Editor displays the following information about the Web service operation file.

View the information, or edit the following fields, if desired.

Web Service Operation File Editor Fields

Field	Description
Web Service Information	
Service	Displays the name of the Web service as defined in the WSDL file.
Operation	Displays which service, or operation, within the Web service this operation uses. The name of this operation is as defined in the WSDL file.

Web Service Operation File Editor Fields (continued)

Field	Description
Style	Displays one of the following values: • **rpc** • **document** • **wrapped** See the W3C Web Services Description Language (WSDL) 1.1 specification.
Use	Displays one of the following values: • **literal** • **encoded** See the W3C Web Services Description Language (WSDL) 1.1 specification.
Beans Package	Displays the name of Java beans package that Dialog Designer uses to implement the Web service.
Name Space	Displays the URL to the location where the Web service resides.
Endpoint	Displays the endpoint used to invoke the Web service. The required endpoint for the Web service is available within the WSDL file for the Web service. **Note:** When deploying the application, this endpoint can be changed to another endpoint, in Configure Web Application Descriptor. This change might be useful, for example, if you have been using a pared down version of the Web service while developing and testing your application. In this case, you might want to use a different, official version of the Web service with the deployed application.
Username Token authentication	When generating the security token to include in the SOAP message for the Web Service request, the password can either be sent as a hash (#), or in the clear. When interfacing to .Net Web Services over a secure link, it can be beneficial to not hash the password, since WSS4J and .Net use different hash schemes.
Timeout	Sets the HTTP socket timeout, in seconds, to be used when executing the Web Service operation.

Web Service Operation File Editor Fields (continued)

Field	Description
WSDL URL	The URL of the WSDL file for the Web service in standard HTTP or HTTPS format. All Web services has an associated WSDL file that describes what services the Web service provides. Dialog Designer uses this WSDL information to populate the **Operation** field list. **Note:** Dialog Designer supports only WSDL files that contain unique method signatures. **Note:** Dialog Designer does not support operation overloading, or the use of methods with the same name but different parameters. **Tip:** If a Web service has been previously created, Dialog Designer remembers the URL. For future Web Service creations, Dialog Designer automatically populates this field with the URL. Dialog Designer keeps track of the last 20 Web services URLs used in operations.
Input Parameters	
Input Parameters	The parameters configured when the Web Service was created is displayed, but this can be changed, if desired. Using these fields, variables can be remapped after a Web Service operation file is initially created. This is useful for when custom variables are created after creating a Web service operation file. See Mapping Input & Output Parameters (Wizard Page 2 & 3) for more information.
Output Parameters	
Output Parameters	The parameters configured when the Web Service was created is displayed, but this can be changed, if desired. Using these fields, variables can be remapped after a Web Service operation file is initially created. This is useful for when custom variables are created after creating a Web service operation file. See Mapping Input & Output Parameters (Wizard Page 2 & 3) for more information.
Authentication	
Authentication	Indicates how Dialog Designer should authenticate with the Web Service host. The authentication configured when the Web Service was created is displayed, but this can be changed, if desired. Options are: • **Basic** – Transmits the username/password pair in an unencrypted form from browser to server in the HTTP header. • **Digest** – Transmits the username/password pair encrypted from the browser to server in the HTTP header.

Web Service Operation File Editor Fields (continued)

Field	Description
Headers	
HTTP Header	.

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 a Dialog 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 DD variable. If the direction is set to OUT then the value of the DD variable is added to a header sent with the web service request.

Employing Web Services in Dialog Designer Applications

Once the Web service operation file has been created, it can be used in call flows. Web service operation files can be used only within a Data node.

To employ a Web service operation file in a call flow:

1. Place a Data node in the Call Flow Editor main workspace.
2. To open the Data node editor, double-click the Data node.
3. From the Data node editor palette, select the **Web Service** item and place it in the editor workspace.
4. In the **Avaya Properties** view, **Name** field, select the database operation file to use.

Re-generating Web Services Client Code

When Dialog Designer interacts with a Web Service, it uses client code generated by Axis. This code is called the Web Service Client. This code is first generated upon creating a Web Service operations file (**.wsop** file) using the Web Service Operations File Wizard.

To regenerate the Web Service client:

1. Right-click and select **Properties** on a created Web Services operations file located in the Navigator panel (go to **[*project*] > connectivity > wsoperations > [**.wsop file*]**).
2. In the **Properties** panel, click the **Generate Client** option in the left column.
3. Click **Generate** in the Generate Client dialog.

   Dialog Designer connects to the Web Service and request the WSDL using the WSDL URL and regenerate the Web Service Client based on the WSDL document returned by the Web Service.

Consuming Web Services over HTTPS

When working with certificates and key stores, Dialog Designer uses the keytool tool included with JDK, and by default installed in **JAVA_HOME\bin**.

To consume a Web service over HTTPS, the server's certificate needs to be installed into a trusted keystore on the application server, and the system property **javax.net.ssl.trustStore** points to the trust store. (The same is true for the ADE and can also be done on the ADE).

To import the server certificate:

Note:
Adjust various paths and passwords to fit your configuration.

1. Get the certificate that is installed on the server.

   In the following it is assumed that you have the server certificate stored in a file called **Server.cer** on the application server.

   If you do not have the server certificate, you can use keytool to export the certificate from the server and into a file by running the following on the app server.

   keytool -export -rfc -alias MyAlias -file Server.cer

2. On the application server, create a new keystore by importing the server certificate:

   keytool -import -trustcacerts -alias MyAlias -file Server.cer -keystore MyKeystore

3. Configure the app servers JVM to use the new keystore, by setting the following JVM parameters:

   javax.net.ssl.trustStore=full path to MyKeystore

   javax.net.ssl.trustStorePassword=the keystore password

4. Restart the app server.

Certificate problems can be very hard to troubleshoot, so if you are having problems it can be helpful to enable more debugging information by setting the following parameter:

javax.net.debug=ssl,handshake,trustmanager

For more details on keytool, see:

http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/keytool.html

For more details on certificates see:

http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html

Chapter 14: Working with Event Types

In the context of Dialog Designer, the term *event* refers to an abnormal or unexpected situation which occurs during application run time. These can include error situations, for example, when the application tries to retrieve a speech resource that is unavailable for some reason. Events can also occur when callers do not respond as expected, either with responses for which the system cannot find a match or by not responding at all, for example.

The W3C VoiceXML Specification and Dialog Designer both include a number of built-in event types that any VoiceXML-compliant server should be able to recognize and handle. Dialog Designer makes it easy for you also to create your own custom event types to use in your applications.

This chapter describes all aspects of working with events and event types. It contains the following sections:

About Events

In general, when a Dialog Designer speech application encounters a situation to which it does not know how to respond, or which meets predefined conditions, then Dialog Designer “throws” an event. The Avaya Application Simulator then searches to locate the appropriate *event handler* to handle, or “catch,” the event. When Dialog Designer locates the appropriate event handler, it responds according to the instructions provided in the event handler.

The following sections describe details about events in more detail:

Event Handlers and Scope

Event handlers can be placed at several different levels within the speech application:

- **In the AppRoot node** - Event handlers placed in the AppRoot node act as global event handlers. That is, they are active and ready to handle events that occur anywhere, at any level, in the application.

  See Setting Global Application Properties: AppRoot Node.

- **In any other node** - Event handlers placed at the top level of any node other than the AppRoot node act as form-level event handlers. That is, they are active and ready to handle events that occur anywhere within the node in which they are placed.
- **In an item within a node** - Event handlers attached to node items within a node act as field-level event handlers. That is, they are active and ready to handle events that occur only within the item to which they are attached.

Event handlers at a lower level take priority over event handlers at a higher level. In other words, suppose you have an event handler for No Match events at the field level, and you have another event handler for No Match items in the AppRoot node. If a No Match event occurs at the field level, the event handler at the field level takes precedence over the event handler in the AppRoot node.

Types of Event Handlers

Dialog Designer offers two basic types of events:

- *Built-in events* are events for which Dialog Designer has incorporated event handlers. Examples of these include the No Input, No Match, and Help events. These events are defined as “default catch elements” by the W3C VoiceXML Specification.

  For more information about these event handlers, see Built-in Event Handlers.

- *Custom events* are events that you define and create handlers for within your Dialog Designer speech applications.

  For more information about creating and handling custom events, see Custom Events.

Built-in Event Handlers

The W3C VoiceXML 2.0 Specification specifies a number of events for which VoiceXML 2.0-compliant applications and browsers are expected to provide implicit event handlers. Dialog Designer complies with this standard, by providing the following built-in event handlers:

Note:
For details on each event handler, click the appropriate link.

- No Input
- No Match
- On Disconnect (exit)

If you do not use these event handlers anywhere, Dialog Designer still employs them on a global basis to handle these types of events with a built-in default response. Use these node items when you want to override the default response or customize the event handling for these types.

In addition, Dialog Designer has built-in event handlers for *maxspeechtimeout* and *connect.disconnect* events, as required by the VoiceXML 2.0 recommendation.

For more information about using built-in events, see Events in Applications.

Custom Events

Custom events are events that you define and create handlers for within your Dialog Designer speech applications. These events can be assigned any name, for example, “myevent.thishappened”. The W3C VoiceXML 2.0 Specification defines various event and error events.

For example, suppose an event handler is created to handle a situation in which the account balance for a caller is below a certain amount. In this case, there might be a node of the call flow in which the account balance is retrieved from a database, and the balance is assigned to a variable. A Data node can be used to compare the balance with a set amount, and if the balance is less than that amount, the node operation can then throw a custom event called **BalanceLow**.

A **Catch** event can then be used in the AppRoot to handle this event when it is thrown from anywhere in the application (several nodes might throw this particular event). The **Catch** event can then be set up to transfer the caller to a live agent to talk about the account balance and offer overdraft protection services, for instance.

Note:
When catching errors, only catch individual specific errors and do not use a “catch all” such as “error.*” or “.”. Catching an error and taking inappropriate actions can lead to unpredictable results.

Use the Event Types Editor to create custom event types. See Using the Event Type Editor.

Once the event type is created, then you can use it in your applications by throwing the event when the situation or condition is met and then catching it elsewhere in the application. For more information about this practice, see Events in Applications.

Try Catch Event Handling

Try/Catch elements in the Data node will *try* to execute items under the Try item. To handle exceptions, add “Catch” items (under the parent Try).

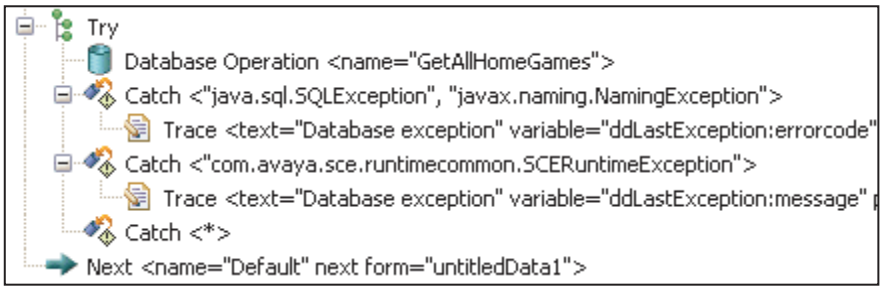
By default, Catch <*> will catch *all* exceptions. Catch can also be used to catch specific exception types for handling specific errors differently. Comma separated exception types can also be specified for catching different exception types, but handling them in the same way.

List of common exceptions are provided in the Properties view. Currently this list contains:

- SQLException
- IOException
- SCERuntimeException

A new variable caller **ddLastException** automatically captures the exception caught by a Catch item (includes the following data members: errorcode, message, object, stacktrace, and type).

Following is an example of the Try/Catch mechanism.



Using the Event Type Editor

Use the Event Type Editor to create custom event types for use in your application. Also see Custom Events.

To create a custom event type:

1. To open the Event Types Editor:
   a. Select the Call Flow Editor as the active workspace.

b. On the main toolbar, click the Event Types Editor icon ().

2. In the palette, click **Event Type**.
3. Click in the workspace.

   Dialog Designer displays a new event in the workspace.
4. Select the new event type item in the workspace.
5. In the **Avaya Properties** view, the **Name** field, enter the name to assign to the new event type.

For information about how to use custom events, see Using and Handling Custom Events.

Events in Applications

As described in general terms in About Events, the basic process for events in applications involves the following steps:

1. The application encounters an unexpected condition or error, known as an *event*.
2. The Avaya Application Simulator identifies the unexpected condition or error and throws an event that matches it, as closely as possible.
3. Starting at the level where the event was thrown, the application works its way up from field to form to global level until it finds an appropriate event handler to catch the event. If no event handler is defined, the application uses one of the implicit (built-in) event handlers.
4. The event handler instructs the application how to handle the event.

Dialog Designer implements this basic process in different ways, depending on what type of event is thrown and whether there is an event handler set to catch the event. Generally, Dialog Designer has two ways of handling events, depending on whether the event is a built-in or a custom event. See:

- Using Built-in Event Handlers for Default Events
- Using and Handling Custom Events

Using Built-in Event Handlers for Default Events

The Avaya Application Simulator (AAS), which is used to run Dialog Designer applications, has built-in event handlers for the “default catch items” as defined and required by the VoiceXML Version 2.0 recommendation. This means that, for the default catch items, you are not absolutely required to use event handlers, because the AAS can handle them automatically.

These built-in event handlers, however, might not be what you want or need, as they are very generic in their responses. As such, they might not respond to a given event the way you want. Therefore, Avaya recommends that, whenever possible, you consider where and how to use the built-in Dialog Designer event handlers for these events:

- Use built-in event handlers in the AppRoot node to catch any occurrences of those events for which you might not have event handlers in place at lower levels. For more information about using the AppRoot node for global settings, see Setting Global Application Properties: AppRoot Node.
- Use built-in event handlers at the node (form) or node item (field) level within a node, to catch and respond in a specific way to those events that might be thrown at those levels.

For example, you might have a generic No Match event handler that says something like: “I’m sorry, I do not understand you.” This might be fine for a global response, but for a particular Prompt and Collect node, you might want a more specific response like, “I’m sorry, that is not one of the options. For checking account information, please say or press ’one’; for savings account information, please say or press ’two’." In this case, you would add a **No Match** event handler to your Prompt and Collect node.

Note that it is not necessary, with the default catch items, to use a **Throw** item to throw the event. This functionality is already programmed into the AAS, and the system knows when to throw these events. All you need do is provide the event handler so the system knows how to respond when the event is thrown.

Using and Handling Custom Events

Using custom event types is different from using built-in event types in two important respects:

- You must create custom event types before you can use them.

  To create custom event types, see Using the Event Type Editor.
- You must specifically use a **Throw** item to throw a custom event and a **Catch** event to handle it.

There are three basic ways to throw a custom event:

- Use a **Link** item in a node and then use the **Throw** item, attached to the **Link** item, to throw the event. See Link.
- Use the **Throw** item within a **Catch** item. See Catch (VXML Events).
- Use a **Throw** item as the only item in a Form node, and then use a condition within a Data node to branch to that Form node.

  For more information about the Form and Data nodes, see Application Items (Basic Nodes).

Once you have set up a way to throw the custom event, you must make sure you have a **Catch** item, within the scope of the **Throw** item, to handle it.

Chapter 15: Application Testing by Simulation

One of the strongest features of Avaya Dialog Designer is the capability Dialog Designer offers for testing and debugging your speech application projects. In Dialog Designer, you can simulate and test virtually every aspect of your speech application project, including its response to error conditions.

This chapter provides descriptions of the various testing and debugging tools in Dialog Designer and the procedures for testing speech application projects. It contains the following sections:

About Testing Applications by Simulation

The simulation and debugging tools in Dialog Designer make it possible for you to test virtually every part of your speech application project before you deploy it to a live IVR system. This means that you can spend less time having to test and debug your speech applications on the target system. It also means less time that the IVR system must be out of operation while you install new applications.

See the following sections:

What Can Be Tested by Simulation

In Dialog Designer, you can simulate the following features and functions of a call flow:

- The calling number (ANI)
- The called number (DNIS)
- DTMF inputs
- ASR inputs, either using a microphone or by typing in a response

- Problems with input recognition, such as No Match and No Input conditions
- Caller hang ups during the call
- Call transfers and all possible transfer results
- Passing variable values from one module to another, when you are testing individual modules
- Confidence level of ASR recognition
- Interaction Center (IC) and Computer Telephony Integration (CTI) connectors, using a connector simulator and connector scripting

Limitations When Testing by Simulation

There are a few functions that you cannot test by simulation in Dialog Designer. These functions include:

- Databases - Dialog Designer does not provide a way to simulate database operations, so if you want to test those, you must test with a real database. You can, however, use a pared down version of a database for development and testing and then change to a run-time version just before you deploy the application.
- Web services - Dialog Designer does not provide a way to simulate Web service operations, so if you want to test those, you must test with a real Web service. You can, however, use a local or a pared down version of a Web service for development and testing and then change to a run-time version just before you deploy the application.

Debugging Features

Dialog Designer provides the following features and tools designed to help you debug applications:

- **Highlighting during simulation** - While the application project is in simulation mode, Dialog Designer displays highlights each node as the Avaya Application Simulator (AAS) processes the nodes. This feature makes it easy to track progress through the application as the simulation progresses.
- **Input tab, AAS progress display** - During application simulation, the AAS presents a step-by-step readout of what is happening as the simulation progresses. The AAS displays this progress in a pane of the **Input** tab. Even after the simulation ends, you can scroll back through this output to analyze what happened during simulation.
- **The VXML Log tab** - This item is similar to the previous item, but the information that the AAS displays on this tab is much more detailed than in the Input tab display. This information is a detailed transcript of the AAS activity during call flow simulation.

- **Debug tracing to output in Console view or trace log file** - If debug tracing is enabled, Dialog Designer directs the output of the debug tracing to two destinations: the Console view display and the trace log file. This output consists primarily of a transcript of the VoiceXML output that is generated while the application is being run. See Trace.
- **Application tracking node with debug options** - Dialog Designer provides a special node, the application *Tracking* node, which was designed to aid in debugging applications. This node makes use of two palette options—the **Trace** item and the **Report** item—to help with application tracking and debugging. See Report and Trace.
- **Scripting of inputs and responses** - For those situations where you do not want or are not able to use the various built-in mechanisms for simulating caller responses, you can create an XML script to tell the application how the caller responds during simulation. See Creating Scripts for Testing.

Using the Avaya Application Simulator to Simulate Applications

To test an application by simulation in Dialog Designer:

1. Generate the project.

   To generate the project, perform one of the following actions:

   - Right-click the project name in the Navigator view. Then, on the pop-up context menu, click **Generate**.
   - Click the project name in the Navigator view. Then, from the **Project** menu, select **Generate** > **Project**.

   Note:
   If Dialog Designer encounters any errors while the project is being generated, the system displays an alert message to let you know. If that happens, cancel the rest of the operation, go fix any problems, and generate the project again. To find out what errors have been generated, you must look at the **Problems** view. For more information about the **Problems** view, see Problems View.

2. (Optional, but recommended) Start Tomcat.

   Note:
   This step is considered optional at this point, because, if you do not start Tomcat manually now, Dialog Designer starts it automatically when you try to simulate an application.

   To start Tomcat, perform one of the following actions:

   - On the main toolbar, click the Tomcat icon ().
   - From the **Tomcat** menu, select **Start Tomcat**.

Note:

If you make changes to your application while Tomcat is running, you must save the application before you attempt to simulate it again. Each time you save the application, Tomcat automatically updates the context to stay in sync with the latest version of the application. Be aware, however, that if your application uses modules, Tomcat does not automatically update module contexts. So, whenever you make changes to such an application, be careful to either:

- Restart Tomcat before you attempt to simulate the application again.

  This option takes longer but is more reliable.

- To manually reload the context, in the Navigator view, right-click the project name, then select **Tomcat project** > **Reload this context**.

  If you receive an error notice with this option, consult your Tomcat documentation to verify that you have correctly set up the Tomcat Manager application.

3. In the **Call** tab of the **Avaya Application Simulator** view, select the project you want to test.
4. Click **Start Call**.

   The AAS checks to see if Tomcat is running. If it is not, Dialog Designer automatically starts Tomcat. The AAS then proceeds to run the application simulation.

During simulation, a wide variety of run-time and events scenarios can be tested. See the following sections for more information:

- Simulating Run-time Scenarios
- Simulating Events

Simulating Run-time Scenarios

During simulation of an application with the Avaya Application Simulator (AAS), you can test a wide variety of run-time scenarios. These scenarios include:

- Caller Inputs and Telephony System Responses - Caller inputs include DTMF and spoken responses to prompts. Telephony system responses include primarily transfer errors.
- Responses from Scripts - You can use automated scripts to simulate two types of responses:
  - Response scripts simulate verbal responses and telephony system responses to various conditions.
  - IC and CTI connector functions simulate IC system and CTI system responses to various types of requests and error conditions.
- Other Inputs - Other inputs include inputs from other modules, database operations, and Web service operations.

Caller Inputs and Telephony System Responses

Usually, a speech application works by prompting a caller for inputs and then interpreting and responding to those inputs. Inputs can be in the form of DTMF (touchtone) key presses or spoken replies. In Dialog Designer, you can simulate those responses when you test applications.

Simulating DTMF Inputs

To simulate DTMF inputs:

During simulation, the AAS prompts for a response in which a DTMF key press is an option. To indicate that the AAS is waiting for a DTMF input, the **Waiting DTMF** indicator in the **Input** tab turns green.

1. Click the key or keys on the DTMF keypad that correspond with the DTMF input you want to submit.

   As you click the keys, the AAS simulates the corresponding DTMF tone. At the same time, the AAS displays the digits in the field below the keypad.

2. Click **Send Digits**.

Note:
The **Send Digits** button is located below the DTMF keypad in the **Input** tab. Depending on your monitor size and resolution, you might have to increase the size of the **Avaya Application Simulator** view to see this button.

Simulate Spoken Inputs with a Microphone

Note:
Before you can use a microphone to simulate ASR input, you must have the Microsoft Speech SDK installed and properly configured. The Microsoft Speech SDK should have been installed during the Dialog Designer installation process. See Configuring the Microsoft Speech SDK for Microphone Input.

To simulate spoken inputs with a microphone:

1. During simulation, the AAS prompts for a response in which a spoken response is an option. To indicate that the AAS is waiting for a spoken response, the **Waiting ASR** indicator in the **Input** tab turns green.
2. With a microphone attached to your computer and turned on, speak the response.
3. When the AAS recognizes the input, Dialog Designer automatically moves on to the next node of the call flow. If the AAS does not recognize the input, the application throws a **No Match** event and the AAS responds with the appropriate No Match prompt.

You can also simulate barge-in using AAS. By default the barge-in feature of the AAS is disabled, but you can enable it be checking the **Enable Barge-in detection** option (go to **Windows > Preferences > Dialog Designer > Avaya Application Simulator** preference screen).

Note:
It is recommended to use a headset when running AAS with barge-in enabled, since a standard PC microphone can easily pick up sounds from the PC speakers.

Simulating Spoken Inputs without a Microphone

To simulate spoken inputs without a microphone:

You need not have a microphone to simulate ASR inputs to an application. The AAS provides a mechanism to type the response you want to use for simulated ASR input and submit it in place of input from a microphone. Note that the AAS does not evaluate the ASR response to see if it is valid. The AAS assumes that whatever ASR response it receives in this manner is valid and treats it accordingly. This fact is one reason it is better to simulate with a microphone, if possible.

During simulation, the AAS prompts for a response in which a spoken response is an option. To indicate that the AAS is waiting for a spoken response, the **Waiting ASR** indicator in the **Input** tab turns green.

1. In the **Recognition** field, type the input that you want to simulate.
2. (Optional) if you want to simulate the confidence level of the ASR server response, enter in the **Confidence** field a number between **0.0** and **1.0**.
3. Click **Send ASR**.

Responses from Scripts

In Dialog Designer, you can use scripts to automate certain kinds of inputs when testing applications by simulation. When using scripts to automate simulations, you can use scripts for two basic purposes:

- To automatically simulate caller responses. See Using Scripts to Simulate Caller Responses.
- To simulate IC and CTI connector functions and responses. See Using Scripts to Simulate IC and CTI Connectors.

For more information about the creation and use of scripts for simulation, see Creating Scripts for Testing.

Other Inputs

With Dialog Designer, you can create speech applications that use a variety of inputs other than caller responses. Among these are inputs from other modules, database queries and other operations, and use of Web services.

Simulating Module Inputs

To simulate module inputs:

When you use modules as part of a speech application project, you might need to provide inputs to that module or receive outputs from that module. When a module expects to receive input from another application or module, the AAS indicates this by activating the **Input Parameters** display pane of the **Call** tab, when you select the module from the **Available Projects** list.

Simulate Database Operation Inputs

To simulate database operation inputs:

Dialog Designer does not support the simulation of database operations. If you want to test database operations, but you do not want to use a full-blown database or your actual run-time database, you can create and use a pared-down development version of the database. Then, before deploying the application, you can change your application to use the run-time version of the database.

Simulating Web Service Operation Inputs

To simulate Web service operation inputs:

Dialog Designer does not support the simulation of Web service operations. If you want to test Web service operations, but you do not want to use a Web service outside your development environment or your actual run-time Web service, you can create and use a pared-down development version of the Web service. Then, before deploying the application, you can change your application to use the run-time version of the Web service.

Simulating Events

Events are used to help determine how speech applications respond when the unexpected happens. For instance, if a caller does not respond when prompted, the system throws a No Input event. Event handlers are used to instruct the system how to respond to such events.

You can also simulate a variety of common events during application simulation. The following list describes these common events and how to simulate them:

- **No Input** - The version of the Avaya Application Simulator (AAS) that Dialog Designer uses does not use timeout settings to determine when a caller has not responded to a prompt. Instead, you must tell the AAS to interpret a period of silence as a No Input event. To do this, click the **No Input** button on the **Input** tab. The AAS then treats the response to the prompt as if the caller did not respond and throws a No Input event.
- **No Match** - The ASR engine that Dialog Designer uses is capable of recognizing much input, but it is not as robust as many third party ASR engines. To deliberately test a No Match situation, in which the response by the caller does not match any expected response, click the **No Match** button on the **Input** tab. The AAS then throws a No Match event.

- **Hang Up** - To simulate a situation in which the caller hangs up before reaching the end of the call flow, click **Hang Up** on the **Input** tab. The AAS then throws a caller disconnect event.
- **Record End** - The AAS cannot detect extended silence at the end of a recording and automatically use record timeout settings to terminate the recording. To simulate a situation in which the system should terminate a recording, based either on an extended silence or on a DTMF key press, click **Record End** on the **Input** tab. The AAS then terminates the recording and proceeds according to the application call flow.
- **Xfer Status** (Transfer status) - This field is active only when the AAS encounters a Bridged Transfer node. Use this drop-down list to select from and simulate the following possible transfer results:
  - **No Answer** - Simulates a situation in which there is no answer from the destination.
  - **Busy** - Simulates a situation in which the destination number is busy.
  - **Disconnect** - Simulates a situation in which the transferred call is disconnected on the far end, usually as a result of the destination party hanging up.
  - **Maxtime** - Simulates a situation in which the transferred call reaches the maximum allowable time for a transferred call and is terminated by the system.

Chapter 16: Application Deployment

Dialog Designer speech or call control applications can be deployed to an application server that the IVR system invokes, or in the case of speech applications, as a reusable module to be used in another Dialog Designer speech project.

This chapter presents information about all aspects of deploying Dialog Designer speech or call control applications.

It contains the following sections:

Deploying the Application

Deploying a Dialog Designer speech application to an IVR system, or call control application to a Voice Portal system is a two-stage process.

1. The first stage involves exporting the application and all its related files to a compressed package, similar to a ZIP file. Depending on where the application will be deployed, this package is a WAR (Web ARchive) file or an EAR (Enterprise ARchive) file. The WAR or EAR file is delivered to the *application server* where the deployed application will run.

   See Using the Export Dialog Designer Project Wizard.

   Note:
   Avaya recommends that you stop Tomcat before exporting an application. (If you do not, a message dialog may appear, that can be ignored; it will not adversely affect the export.)

2. The second stage involves unpacking and installing the WAR or EAR file on the application server. When this is done and the IVR system is configured to run the application, the application is successfully deployed.

   See Installing Required Files on the Application Server.

Dialog Designer also supports another type of deployment: speech projects that can be used as reusable modules in other speech applications.

See Deploying Projects as Dialog Designer Modules.

Using the Export Dialog Designer Project Wizard

For deployment, the speech application project must first be exported to a WAR (Web ARchive) or EAR (Enterprise ARchive) file using the Export Dialog Designer Project Wizard.

Note:
A WAR or EAR 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. IBM WebSphere and WebSphere Express servlet engines use the EAR file format.

Note:
Avaya recommends that you stop Tomcat before exporting an application.

To access the **Export** wizard, perform one of the following actions:

- From the **File** menu, select **Export...**
- Right-click anywhere in the **Navigator** view, then select **Export...**

Dialog Designer displays the Export Dialog Designer Project Wizard. The following sections describe the successive pages of the Export Dialog Designer Project Wizard:

- Select a Project Page
- Specify Export Parameters Page
- Specify Platform Details Page
- Specify Deployment Parameters
- Configure Web Application Descriptor

Select a Project Page

In the opening Select page of the Export Dialog Designer Project Wizard, the type of export to be performed needs to be designated, as follows:

1. Within the **Select an export destination** panel, navigate to **Avaya Speech Development > Export Dialog Designer Project**.
2. Click **Next** to continue. The Specify Export Parameters Page is displayed.

Specify Export Parameters Page

In the Specify Export Parameters page of the Export Dialog Designer Project Wizard, a project and destination directory for the exported project archive file needs to be designated, as follows:

1. In the **Available Projects** panel, select the project to be to be exported (and ultimately deployed).

   Note:
   Only one project at a time can be deployed, so, if the speech application relies on reusable modules, each module used by the application must be deployed separately.

2. Specify a the **Destination Directory** by either entering a directory path, or clicking **Browse** to navigate to the appropriate path.
3. Click **Next** to continue. The Specify Platform Details Page is displayed.

Specify Platform Details Page

In the Specify Platform Details page of the Export Dialog Designer Project Wizard, target platform settings are optionally configured.

When done (or to skip this page and accept the default settings), click **Next**. The Specify Deployment Parameters are displayed.

Export Dialog Designer Project Wizard—Specify Platform Details Page

Setting	Description
Target Platform	
Platform	Specifies the platform for the run-time version of the loaded resource files. Options are: ● **IR** - Application is to run on an Avaya IR system. ● **Voice Portal** - Application is to run on an Avaya Voice Portal system. ● **Desktop** - (Default) Use this option while you or another developer plan are doing development work on the project, using another computer. ● **Other** - Application is to run on another VoiceXML 2.0-compliant IVR system.
Servlet container	Servlet container options for the IVR system. Options are: ● **Apache Tomcat** - (Default) IVR system uses Apache Tomcat application server software. ● **IBM WebSphere** - IVR system uses IBM WebSphere or WebSphere Express application server software. ● **BEA WebLogic** - IVR system uses BEA WebLogic application server software.
Speech **Note**: The following parameters apply to exports of speech application projects only, not call control application projects.	
Grammar compatibility	Type of ASR server engine to target ASR grammars. Options are: ● **IBM** - Grammars are optimized to work with IBM WebSphere or WebSphere Express ASR engines. ● **Nuance 8.5** - Grammars are optimized to work with Nuance 8.5 ASR engines. ● **Nuance OSR** - Grammars are optimized to work with Nuance OpenSpeech Recognition products (including Quantum). ● **SRGS** - (Default) Grammars conform to the SRGS Version 1.0 standard. Dialog Designer, by default, creates grammars to support all ASR engines. The grammars used at run-time is based on what is selected in this field, and ultimately the value of the following variable in the **web.xml** file: **runtime-asr**.

Export Dialog Designer Project Wizard—Specify Platform Details Page (continued)

Setting	Description
Grammar Caching	Grammar caching option to use. Options are: ● **none** - No grammar caching is permitted for this application. ● **default** - The application uses the IVR system default setting for grammar caching.
Enable Speech Synthesis Markup Language (SSML) generation in project prompts	To enable SSML markers to work in prompts, select this checkbox. To disable SSML markers, clear this checkbox. If you select this check box, when the voice browser is generating the TTS content, it also generates the SSML data to help with the rendering of that content. **Note:** If you select this check box, the AAS generates the SSML data, even in simulation mode during testing. The Microsoft TTS engine that Dialog Designer uses, however, is not capable of generating or using SSML data, so it is not apparent during testing that it is being generated.

Specify Deployment Parameters

In the Specify Deployment Parameters page of the Export Dialog Designer Project Wizard, options that affect the way the project is deployed are configured.

When done (or to skip this page and accept the default settings), click **Next**. The Configure Web Application Descriptor are displayed.

Export Dialog Designer Project Wizard—Specify Deployment Parameters Page

Option	Description
Actions	
Regenerate applications	Check this to have the selected application projects regenerated as part of the export process packaging. By default, this option is not checked. However, if the grammars are retargetted, Dialog Designer automatically sets this option so that the grammars are generated properly within the application.
Rebuild applications	Check this to have the selected application projects rebuilt as part of the export process packaging. By default, this option is not checked. However, if the grammars are retargetted, Dialog Designer automatically sets this option so that the grammars are built properly within the application.

Export Dialog Designer Project Wizard—Specify Deployment Parameters Page

Option	Description
Packaging	
Deploy project class files as	Select how Dialog Designer should deploy the class files for the application project. Select from the following options: ● **JAR file** - (Default) Packages the project .class files into a single, self-contained Java ARchive (JAR) file. ● **class files** - Deploys the project .class files as separate files within the WAR file. For run time, it makes little difference which option is chosen. The **JAR file** option, however, has the following advantages: ● JAR files tend to keep the run-time environment cleaner. ● It is easier to replace a single JAR file, if you need to, than to replace a whole batch of .class files.
Include source code	To make the Dialog Designer source code available for others to study, use, modify, or debug, select this option.
Include project meta files	Project meta files are files such as .prompt files, .gram files, .flow files, and so on. To include the project meta files in the WAR file, select this option. Including these files in the WAR file makes it easier to check files into a source control system, email to others, or deploy the project with all files.
Include extra files and folders	If a designer has included extra files and folders in the /data directory of their project structure, by selecting this checkbox, the files are added with maintained folder structure to the exported .war/.ear file. The resources are copied into the root of the .war/.ear file and directory structures are maintained.
Exclude languages	**Note**: This setting is available for exports of speech application projects only. Select this option to exclude one or more sets of language files currently included in your speech project files in the exported WAR/EAR file. If no languages are included in the project already, this checkbox is not selectable.

Export Dialog Designer Project Wizard—Specify Deployment Parameters Page

Option	Description
Reusable Modules **Note**: This setting is available to exports of speech application projects only.	
Deploy projects to the Dialog Designer reusable modules directory	If the selected projects are designed to be used as reusable modules with other speech application projects, select this option. When selected, the wizard deploys a copy of the WAR file to the reusable modules directory. For more information about this option, see Deploying Projects as Dialog Designer Modules.
Runtime Support	
Create the runtime support zip file for deployment	If deploying a Dialog Designer application for the first time, select this checkbox to generate a zip file that contains all required library files to be deployed on the application server. The created **runtimesupport*platform*.zip** file is generated in the same directory as the .war/.ear file.
Create the runtime configuration application	If deploying a Dialog Designer application for the first time, select this checkbox to create a runtime configuration application. The created **runtimeconfig** .war/.ear file is generated in the same directory as the .war/.ear file. **Important!** Do not run the runtimeconfig on your ADE.
Tracing **Note**: This setting is available to exports of speech application projects only.	
Turn off tracing in deployed application	By default, tracing is off in a deployed application, but uncheck this option to turn it on (which is effectively enabling the **localapptrace** parameter in the **ddrt.properties** file). **Note**: Setting the **ddrt.properties** file for a module has no affect on that module's tracing. The Parent app with module/s will also log trace and report information in its log for the modules as well as itself, but nothing will get logged in the module's trace or report logs.

Configure Web Application Descriptor

The Configure Web Application Descriptor page of the Export Dialog Designer Project Wizard is used to review all previously defined export settings and, if necessary, edit them. Two tabs are included, as follows:

- **Application tab** - Provides a review and editable interface for the application settings that affect the way the project is exported. Most of the settings displayed in the **Parameters** table can be modified in one of the previous pages of the Export Dialog Designer Project Wizard. They can also be modified directly on this page.

  The settings and options that can be modified on this page are the same as those found in the Web Descriptor tab of the **Properties for *ProjectName*** dialog box. See Application Tab.

- **Servlets tab** - Provides a review and editable interface for the various parameters and settings of the deployment Java servlets. In nearly all cases, changes to these settings are not needed.

  CAUTION:
  Avaya recommends that changes are not made to these settings. Changing these settings could have an adverse effect to your application.

When done reviewing or editing these settings, click **Finish**. The Export Dialog Designer Project Wizard packages and exports the selected project to the designated destination based on the following rules:

- If the target destination is a Tomcat application server or WebLogic application server, Dialog Designer generates a WAR file named ***ProjectName.war***, where *ProjectName* is the name of the application project.
- If the target destination is an IBM WebSphere or WebSphere Express application server, Dialog Designer generates a .ear file names ***ProjectName.ear,*** where *ProjectName* is the name of the application project.

Before the speech application can be considered deployed, the project files must be installed on the destination application server. See Installing Required Files on the Application Server.

Installing Required Files on the Application Server

Before you can install the Dialog Designer speech application on the application server, you must export the application project. For information about exporting the project, see Using the Export Dialog Designer Project Wizard.

This section includes the following topics:

- Requirements for the Application Server
- Configuring the Application Server to Use a Proxy Server
- Installing Project Files on Tomcat Servers
- Installing Project Files on WebSphere Servers
- Installing Project Files on WebLogic Servers

- Installing the Runtime Support Files
- Installing IC or CTIC

Requirements for the Application Server

Project files must be installed on a server that meets the following requirements:

- **Operating system** – Dialog Designer applications can run within the following operating systems:
  - Windows XP, any version, with Service Pack 2
  - Windows 2003 Standard or Enterprise Server
  - RedHat Linux, ES 4.0
  - Solaris 9
- **Application server software** – Dialog Designer applications must run with one of the following servlet containers:
  - Apache Tomcat 5.0, release 28 or later or Tomcat 5.5
  - IBM WebSphere or WebSphere Express 6.1
  - BEA WebLogic 9.2

Note:
If you use Tomcat 5.0 to develop, you can only deploy to Tomcat 5.0 (or WL 9.2 or WAS 6.1). If you use Tomcat 5.5 to develop, you can only deploy to Tomcat 5.5 (or WL 9.2 or WAS 6.1).

Configuring the Application Server to Use a Proxy Server

To configure WebSphere or WebLogic as a proxy server:

1. Configure the proxy, as follows:
   - If you are using an HTTP proxy server, set up the **http.proxyHost**, **http.proxyPort**, and **http.nonProxyHosts** for the Java Virtual Machine.
   - If also using a HTTPS proxy server, set up the **https.proxyHost**, **https.proxyPort**, and **https.nonProxyHosts** for the Java Virtual Machine.
2. Configure Tomcat, as follows:
   - On Tomcat, HTTP and HTTPS proxy parameters are set up through the Dialog Designer ddadmin application. See Configuring the Dialog Designer Admin (ddadmin) Web Application for more details.
3. Configure the proxy server, as follows:

- For WebSphere proxy server:
  a. To set the WebSphere Console proxy parameters, go to **Application servers > server1 > Process Definition > Java Virtual Machine > Custom Properties**.
  b. Add a property for each proxy setting.
- For WebLogic proxy server:
  a. To set the WebLogic Server Administration Console proxy parameters, go to **Home > Summary of Servers > examplesServer**.
  b. Access the Server Start tab to add arguments to the server.
  c. Parse arguments to the Java Virtual Machine using the *-jvmargs* parameter.

Installing Project Files on Tomcat Servers

To install project files on an Apache Tomcat application server:

1. Copy the WAR file to the **webapps** directory.

   If you installed Tomcat to the default directory, the path for this directory is:

   C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps

   or

   C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps
2. Start, or restart, the Tomcat server engine.

   When Tomcat is started, the application should be available and functional.

Note:
Before deploying speech applications, developers need to understand how to configure database connection pooling (DBCP) to the Web server. The following link provides some useful information of the values supplied, but which need to be tuned for each installation:

http://jakarta.apache.org/commons/dbcp/configuration.html

Installing Project Files on WebSphere Servers

To install project files on an IBM WebSphere or WebSphere Express application server, use the WebSphere Administrative Console as follows:

Note:
The following procedure is intended as a general guideline. Avaya assumes that you are familiar with the WebSphere Administrative Console and how to use it, that you know when the software must be stopped and restarted, and so on. For detailed information about using the IBM WebSphere Administrative Console, see the IBM WebSphere documentation.

1. Open the WebSphere Administrative Console.
2. From the **Applications** menu, select **Install New Application**.
3. Use the browse window to locate the ***ProjectName.ear*** file you copied to the server.
4. If your application uses a JDBC data source, define the data source:
   - For the JDBC data source name, enter the name of your JDBC data source.
   - For the JNDI name, enter **jdbc/**, followed by the name of your JDBC data source.

   For example, if the name of your JDBC data source is **MyJDBC**:
   - For the JDBC data source, enter **MyJDBC**
   - For the JNDI name, enter **jdbc/MyJDBC**

   Note:
   If the JNDI name is not properly defined, your Dialog Designer application will probably fail to access the data source properly.

   
   Tip:
   To determine the name of your JDBC data source, if you do not know it:
   - From the **Project** menu, select **Properties**.
   - In the **Properties** dialog box, select **Dialog Designer**.
   - Click the **Data Sources** tab.

   The JDBC data sources are listed on the **Database** sub-tab.
5. For the rest of the options, accept the defaults.
6. Once the application is finished installing, save your changes and start the application.

Installing Project Files on WebLogic Servers

Use the WebLogic Administrative Console to first set up JDBC data sources, and then deploy the project files on an BEA WebLogic application server. See the

Note:
The following procedure is intended as a general guideline. Avaya assumes that you are familiar with the WebLogic Administrative Console and how to use it, that you know when the software must be stopped and restarted, and so on. For detailed information about using the WebLogic Administrative Console, see the BEA WebLogic documentation.

See the following sections for more information:

- Setting up JDBC Data Sources
- Deploying to WebLogic 9.1

Setting up JDBC Data Sources

To set up JDBC Data sources:

1. Click **Lock & Edit** to edit the configuration.
2. On the left hand navigation menu, choose **Services > JDBC > Data Sources**.
3. Click **New** to create new data source.
4. Fill out the following data source details:
   - **Name** – Name the data source a logical name.
   - **JNDI Name** – jdbc/NAME_OF_DATASOURCE (the current tip works fine to determine name)
   - **Database Type** – Choose your database from list.
   - **Driver** – Choose the correct driver for your database.

   Move to the next screen.
5. Accept the default settings and move to the next screen.
6. Fill out the following information for the connection properties:
   - **Database name** – The name of your database.
   - **Hostname** – The name of the computer hosting the database.
   - **Post** – The port the database is listening on.
   - **Database username** – Username for the database.
   - **Password** – Password for the username.

   Move to the next screen.

7. Click **Test** to test the database
8. Click **Finish**.
9. Click **Activate Changes** to save all changes.

Deploying to WebLogic 9.1

To deploy the application files to WebLogic 9.1:

1. Click **Lock & Edit** to edit the configuration.
2. On the left hand navigation menu, select **Deployments**.
3. Click **Install**.
4. Navigate to your application.
5. Accept the default settings and move to the next screen.
6. Under the **Source Accessibility**, select **I will make the deployment accessible from the following location** and verify that the input box contains the path to your application.
7. Click **Finish**.
8. Click **Activate Changes** to save all changes.

Installing the Runtime Support Files

For each application server where Dialog Designer applications will be run, run-time support files must be installed. To install the run-time support files:

1. Locate the appropriate run-time support file for your application server. They are:

   runtimesupportTomcat.zip

   runtimesupportWebsphere.zip

   runtimesupportWeblogic.zip

   runtimesupport*platform*.zip is generated when the **Create the runtime support zip file for deployment** checkbox is checked in Export wizard. It is located in the same place as the .war/.ear file. See Using the Export Dialog Designer Project Wizard.

2. Extract, or unzip, the run-time support files to one of the following locations:

   - For an IBM Websphere or WebSphere Express run-time environment:

     ***WebSphereHome*/AppServer/**

     where *WebSphereHome* is the directory in which you installed the IBM WebSphere servlet engine software.

- For an Apache Tomcat run-time environment:

  ***TomcatHome*\common**

  where *TomcatHome* is the directory in which you installed the Tomcat servlet engine software.

- For a WebLogic run-time environment:

  bea\user_projects\domains*domainname*

The following table describes the files included in the run-time support files:

Filename	Description
scertcommon-*versionNumber*.jar	The shared Dialog Designer run-time classes. *versionNumber* represents the version of Dialog Designer. **WARNING:** If updating a previous installation of Dialog Designer, all previous instances of this file must be deleted before running Dialog Designer.
weblm.jar	The client library for license management.
activation.jar	The Java beans activation framework.
commons-codec-1.3.jar	The classes for encoding and encrypting data.
commons-discovery-0.2.jar	The classes for discovering and loading pluggable services (for example, logging, XML parsing, and so on).
commons-logging-1.0.4.jar	Logging component used by Web services.
opensaml-1.1.jar	Security component used by Web services.
VPApplogClient.jar	Used for writing logs to Voice Portal.
xmlsec-1.3.0.jar	XML component used by Web services.
log4j.properties	Used for configuring the logs.
tsapi.pro	Used for configuring the CTI component.
log4j-1.2.8.jar	The classes for log4j logging activities.
axis.jar	Used for Web services operations.
jaxrpc.jar	Used for Web services operations.
saaj.jar	Used for Web services operations.
wsdl4j-1.5.1.jar	Used for Web services operations.

Installing IC or CTIC

See the following instructions for installing IC or CTIC to your production deployment system:

- Installing IC
- Installing CTIC

Installing IC

The ICConnector should be manually deployed to your production system if you have an application referencing the ICC.

The necessary files can be found in

eclipse\plugins\com.avaya.sce.ic.ui\data

icconnector.war

icconnector.ear

In addition, the following installation steps are required:

- For Tomcat:
  - Copy the icconnector.war to your webapps.
  - When you restart Tomcat, it will expand and deploy for you.
- For Websphere:

  Use the **icconnector.ear** file and deploy it using the Websphere Admin Console.
- For WebLogic:

  Use the icconnector.ear (preferred) or .war file and deploy it using the WebLogic Administrative Console.

Once the ICC is deployed, you will need to configure it using the Dialog Designer Runtime Admin Configuration. See Configuring the Dialog Designer Admin (ddadmin) Web Application.

Installing CTIC

The CTIConnector should be manually deployed to your production system if you have an application referencing the CTIC.

The necessary files can be found in

eclipse\plugins\com.avaya.sce.cti.ui\data

cticonnector.war

cticonnector.ear

In addition, the following installation steps are required:

- For Tomcat:
  - Copy the **cticonnector.war** to your webapps.
  - When you restart Tomcat, it will expand and deploy for you.
- For Websphere:

  Use the **cticonnector.ear** file and deploy it using the Websphere Admin Console.
- For WebLogic:

  Use the **cticonnector.ear** (preferred) or **.war** file and deploy it using the WebLogic Administrative Console.

Once the CTIC is deployed, you will need to configure it using the DD Runtime Admin Configuration. See Configuring the Dialog Designer Admin (ddadmin) Web Application.

Preparing the Application Server to Run Dialog Designer Applications

Before a Dialog Designer application can be run on an application server, the application server where the Dialog Designer application will be run must be prepared. This preparation consists of:

- Configuring the Dialog Designer Admin (ddadmin) Web Application
- Configuring the IVR System to Use the Speech Application
- Running a Dialog Designer Application under SSL Control

Configuring the Dialog Designer Admin (ddadmin) Web Application

In previous releases of Dialog Designer, application server configuration items were configured in an application's **web.xml** file. With Dialog Designer 4.0 and beyond, common application server configuration are centrally configured within a Web-based configuration module—the Dialog Designer Admin Console.

See the following sections:

- Accessing the Dialog Designer Admin Console
- Configuring License Server
- Configuring Proxy Server Settings
- Configuring CTI Settings

- Configuring IR Channel Map
- Configuring IC Setting
- Configuring Users

Accessing the Dialog Designer Admin Console

In previous releases of Dialog Designer, application server configuration items were configured in an application's **web.xml** file. With Dialog Designer 4.0, common application server configuration are centrally configured with a Web-based configuration module—the Dialog Designer Admin Console.

To access the Dialog Designer Admin Console (often referred to as ddadmin):

1. With Eclipse open, start Tomcat.
2. Within a Web Browser, access the Dialog Designer Admin Console at the following URL, typically:

   http://localhost:8080/runtimeconfig
3. When first accessing this URL, the Login page is displayed. Enter the default username and password pair: **ddadmin/ddadmin**.

   Upon first logging in, a Dialog Designer "Legal Notes" is displayed.

Note:
If the Login page does not display, or the default username/password does not work, contact Avaya Dialog Designer support.

Also displayed are links to configuration pages.

See the following sections:

- Configuring License Server
- Configuring Proxy Server Settings
- Configuring CTI Settings
- Configuring IR Channel Map
- Configuring IC Setting
- Configuring Users

Note:
Some of the configurable settings may seem redundant with the configured settings in Dialog Designer preference settings. Though the types of settings are indeed redundant, they are not literally redundant. Preference settings are configured for the development environment. These settings are to be configured for the deployed application server hosting your deployed applications.

Configuring License Server

From within the Dialog Designer Admin Console, enter the URL to the Weblm license server host. The same URL is used for all applications on the application server. Also, a “path” if not required.

For example: *http://myhost:8080/*

Configuring Proxy Server Settings

Proxy settings are required to be configured for the application server. Set both the Non-secure and secure proxy HTTP settings, as well as non-proxy hosts. Configure these settings as described in the following table.

Refer to Dialog Designer release notes for specific issues related to Websphere and WebLogic with regards to proxy server settings, as applicable.

Use Proxy	Flag to indicate that the non-secure proxy settings should be used.
Proxy Host	Full HTTP path to, or URL of, the proxy server.
Proxy Port	Port that Dialog Designer can use to access the proxy server.
Non Proxy Hosts	Names of hosts that do not need to be accessed through the proxy.
Use Http for Https	Option to quickly populate the “secure” proxy settings with same values indicated in the non-secure proxy settings.
Secure Use Proxy	Flag to indicate that the secure proxy settings should be used.
Secure Proxy Host	Full HTTPS path to, or URL of, the proxy server.
Secure Proxy Port	Port that Dialog Designer can use to access the HTTPS proxy server.
Secure Non Proxy Hosts	Names of secure hosts that do not need to be accessed through the proxy.

Configuring CTI Settings

If the Dialog Designer application uses the Compute Telephony Integration (CTI) connector, the application server must be configured to communicate with the TServer/AES. Configure these settings as described in the following table.

Once a server has been added, the entry is added to the CTI server table, and links are added to configure channel extension mappings or to add a failover server. When clicking **map**, a new page is displayed to configure TServer extension mappings. Channels can be mapped one at a time, or a sequence of channels can be mapped using a "1-n" reference. For IR, channels must map to IR setup. For Voice Portal, channel mappings are arbitrary. (For this release, ignore **Observe On Setup** option.)

If one already exists, the **cti_configuration.xml** file can be uploaded to populate CTI settings. Ensure that the cti_configuration.xml file has proper format.

Note:

You will need to restart the CTI Connector for changes to take affect. You will also need to modify **tsapi.pro** that is included with your runtime support files before connecting to a TServer/AES.

Copy or move the **tsapi.pro** file from **bea\user_projects\domains\ domainname\ lib** to **WeblogicHome\common\lib**. You can then edit **tsapi.pro** file in this directory as well.

Tip:

A delay could be incurred when the CTI connectors are attempting to establish connections to TServers, which can slow down the Tomcat launch as well. When connectors are not installed, initialization is quicker. If this is problematic, it may help to use "localhost" in the tsapi.pro file instead.

See About CTI Connectors.

Attributes	Description
General Settings	
Timeout	Time in ms to wait for a VOX response. Default is 4000.
Trace Verbosity	Amount of debug output. Possible values are: • 0-off • 3 full

Attributes	Description
CTI Settings	
Name	A unique server name to identify the TServer. Once added, the table list provides a link to configure channel-extension mapping, as well as a failover server (in the event that the primary server goes down). Channel-extension mapping is the same as the primary for the failover server, so these configurations are not required also for the failover server. **Note**: Server name must be unique and the TServer/AES and failover names cannot contain '*'. TServer/AES names cannot be duplicated (even Failover name cannot be the same as TServer/AES).
Service Name	Identifies the service provider in the following format: vendor#switch#type#server For example: AVAYA#S8300DD#CSTA#MLDDAES
User Name	Username to connect to this tserver/AES.
Password	Unencrypted password to connect to this tserver/AES.
Confirm Password	Confirm password must match password.

Configuring IR Channel Map

The IR Channel Map configuration page is where virtual channel mapping for IR systems is defined. IR systems can overlap their physical channels. This mapping changes a physical channel to a virtual channel so that two or more IR systems can co-exist without overlap.

This configuration is necessary for both IC and CTI connectors.

Enter a **Call tag** to identify an Avaya IR system, channel number and timestamp. The format to be entered is: ***Avaya IR calltag Format: machine_name channel_number timestamp***.

Call tags are specific for each IR system and is passed into the application. Use only the first part of the call tag—machine name—to identify the system.

Once a Call tag is added, it is added to the list, and a **VChannel Map** link is available. Click this link to enter the physical channel and virtual channel for this mapping. (All IR systems have the same physical channel. Virtual channels should not overlap.

Tip:
Leave one system straight 1:1 mapping, change the rest.

If a virtual channel map already exists, browse to it on your local system and upload the ***irvirtualchannelmap.xml*** file.

Configuring IC Setting

If the Dialog Designer application uses the Interaction Center (IC) connector, the application server must be configured to make the connection with the VOX server in the IC system.

IC Channel Mapping is used to inform ICC which VOX to target the call to. This mapping is only necessary in client mode *and* if you have more than one VOX connection. Enter the channels, and the ICC match the channel to the correct VOX and send the call to it.

For more information, and the procedure to configure the application server to use the IC connector, see About the IC Connector.

Attributes	**Description**
General Settings	
Timeout	Time in ms to wait for a VOX response. Default is 4000.
Trace Verbosity	Amount of debug output. Possible values are: • 0-off • 3 full
IC Settings	
Name	Unique name to identify the IC
VOX Address	Address of VOX Server. Only applicable if ICC initiate the connection to VOX
Port	TCP/IP port

Configuring Users

ddadmin is the default configured user. Additional users can be added for managing or configuring the settings in the Dialog Designer Admin Console by adding them to this page.

To add a user, enter a username, password, and confirm password, then click **Add**. The new user is added to the list of users. (To delete a user, select it and click **Delete**. To change a user's password, click the associated **change** link.)

Configuring the IVR System to Use the Speech Application

The procedures to configure a particular IVR system to use the speech application depends on the IVR system itself. The Avaya Voice Portal system, for example, requires you to administer and configure the application in the Voice Portal Management System.

For more information and the procedures required for your system, see the documentation for your IVR system.

Running a Dialog Designer Application under SSL Control

To deploy your Dialog Designer applications to work under SSL control, configure your application to run under SSL control as follows:

- Use the https://<*MyServer*>/<*MyDDApp*>:8443:/<*StartURL*>
- Add a security constraint in the Dialog Designer application **web.xml** file so that either http:// or https:// can be used. Tomcat forces SSL when it is run.

In addition:

- If you are using VoicePortal 3.0 and Nuance OSR, you must modify your Dialog Designer applications to use dynamic grammars if Nuance OSR has an issue with static/external grammars.
- If the parent Dialog Designer application is under SSL control, you must also put all of its child applications under SSL control as well. This includes modules, CTICs, and ICs.

Hot Deployment: Updating Applications without Bringing the System Down

Avaya recommends that developers always attempt to design and create highly available Web applications, or Web applications that can be deployed or updated without bring the system down. This is referred to as “hot deployments”. Typically, this is achieved using hardware or software solutions to achieve application updates utilizing multiple servers and a load balancer.

Hot deployments essentially use HTTP routing (load balancer or IP sprayer) to route requests away from the server that will be upgraded. With sticky sessions (or session affinity), all active sessions on the server are allowed to finish (or “drain”) while other servers assume the load of the server that is being taken down. When all sessions have been drained, the application on the server can be stopped and the update applied

In such a case, the server does not actually have to stop unless common libraries are updated. This means that other applications on the server can continue to run uninterrupted.

After updating the application, it can be restarted and then traffic can be gradually routed to it to make sure that it is running properly. This approach also allows for updates to be backed out if there are problems with the new application by routing requests back to the servers hosting the older application version.

Hot deployment approaches for developing Web application updates while maintaining application availability is widely documented on the Web for deployment to enterprise-level application servers.

Following are a handful of references, but the Web has many more:

- "High Availability Tomcat"

  http://www.javaworld.com/javaworld/jw-12-2004/jw-1220-tomcat.html

- "Maintain continuous application availability while updating WebSphere Application Server enterprise applications"

  http://www-128.ibm.com/developerworks/websphere/techjournal/0412_vansickel/0412_vansickel.html

- "Understanding WebLogic Integration High Availability"

  http://e-docs.bea.com/wli/docs81/deploy/highav.html

Deploying Projects as Dialog Designer Modules

Note:
This section only applies to speech applications.

When creating a large and complex speech application, it is often a good idea to identify parts of the application that can be reused by other parts of the application. Reusable parts can be created as modules.

For example, in a banking application, a module might collect the account number of a caller and then write the account number to a variable. This module may be used by many different parts of the main application, including the parts devoted to checking accounts, savings accounts, mortgage accounts, and so on.

To deploy a Dialog Designer Speech Project as a reusable module:

1. Open the **Export** wizard by performing one of the following actions:
   - From the **File** menu, select **Export...**
   - Right-click anywhere in the Navigator view, then select **Export...**

   Dialog Designer displays the **Export** wizard, **Select** page.

2. From the **Select an export destination** list, select **Export Dialog Designer Projects**, and then click **Next**.

   Dialog Designer displays the **Export Dialog Designer projects** page.

3. From the **Available Projects** list, select the project you want to export as a reusable module, and then click **Next**.

   Dialog Designer displays the **Configure web applications descriptors** page.

   For information about the settings on this page, see Configure Web Application Descriptor.

4. Make any changes you need to on this page, and then click **Next**.

   Dialog Designer displays the **Options** page.

5. Select the check box labeled: **Deploy projects to the Dialog Designer reusable modules directory**

   Dialog Designer displays a warning with notification that after the module is deployed to the reusable module directory, there will be both a reusable module *and* a speech project, each with the same name in the project workspace. To avoid potential conflicts between the speech project and the reusable module, close the speech project and move it out of the workspace. Then, if additional modifications need to be made to the speech project and it needs to be redeployed later, it can be moved back into the workspace.

   Even if the project is left in the workspace, be aware that the deployed reusable module, (available as a module node from the Call Flow Editor palette) and the project by the same name are in fact two different things.

6. In the **Deploy Project** message box, click **Yes**.

7. Click **Finish**.

   Dialog Designer deploys a copy of the speech project to the reusable modules directory and places a module item for the new module on the Call Flow Editor palette.

   This module can be used as a module node in speech applications. For more information about module nodes, see Module Nodes.

Note:

The Export wizard also creates a copy of the WAR file at whatever location you specified as the **Destination Directory** on the **Export Dialog Designer projects** page. This WAR file is the file to deploy on the application server for use with the parent application. If you do not deploy this to the application server along with any other application project WAR files, the parent application will not be able to find or use it.

Note:

If you are storing "custom object" (objects that you define) in session properties and wish to pass them between to modules (Web applications), the code for the objects must be in the **<tomcat_home>/common/lib** so that the class loader works properly.

If that code is on **<appname>/WEB-INF/lib** then an object instantiated in project A cannot be accessed on project B. By moving the implementation of the class to the common lib and in turn the common class loader, the implementation may be shared across Web applications.

About the Application Execution Environment

At run time, the IVR system starts the speech application on the application server. To call the application and start it running, you must provide your IVR system with the URL to the start page for the application. This URL should follow the format:

http://*myAppServer.myCompany.com:port*/*appName*/Start

where:

- ***myAppServer.myCompany.com*** is the fully qualified domain name or IP address of your application server.
- ***port*** is the port used to access the application. The default for this varies according to the application server:

For application servers:	Default port:
Tomcat	8080
IBM WebSphere	9080
IBM WebSphere Express	7080
BEA WebLogic	7001

- ***appName*** is the file name of the application you want to run.

During application execution, the speech application directs the call flow and issues requests for additional resources.

The application server for the IVR system, in turn:

- Controls execution of the generated code and subsequent generation of the VoiceXML pages.

- Uses the servlet engine to serve the VoiceXML pages to the appropriate component within the IVR system.
- If appropriate, uses SOAP and WSDL to access and use Web services.
- If set up to do so, the IVR system accepts application data, generates trace and report data, and writes the data to the appropriate files on the IVR system or application server.

The deployment process also provides tools that allow you to check various aspects of the application in the run-time environment. See Checking the Application Deployment.

Checking the Application Deployment

If your application is not behaving as expected in the deployment environment, Dialog Designer offers a number of utilities to help you debug and figure out what is going on. You can access these utilities with a Web browser.

Note:
These utilities are generated during the application deployment process, so they are only available to applications that you have actually deployed, not in the Dialog Designer application development environment.

See the following sections:

- Accessing the Application Deployment Utilities
- Validating the Application Deployment
- Viewing the Application in HTML Mode
- Monitoring Application Performance
- Viewing Application Trace and Report Logs

Accessing the Application Deployment Utilities

The procedure to access the application deployment utilities is slightly different, depending on whether you deployed your application to a Tomcat or an IBM WebSphere/WebSphere Express application server.

See the following sections:

- Accessing Utilities on a Tomcat Application Server
- Accessing Utilities on an Application Server

Accessing Utilities on a Tomcat Application Server

To access these utilities on a Tomcat application server:

1. Open a Web browser window.
2. In the Address field, enter the following URL:

   http://*domain.name*:8080/

   where ***domain.name*** is the fully qualified domain name of the Tomcat server.

   Tip:
   As a shortcut, or if you do not want to use the Tomcat Manager, you can enter the following URL in this step: **http://*domain.name*:8080/*appName***, where ***appName*** is the name of the application you want to check on.
3. Click **Tomcat Manager**.
4. Click the name of the application you want to check on.

   The Web browser displays the index.html page for the selected application.

Accessing Utilities on an Application Server

To access these utilities on an application server:

1. Open a Web browser window.
2. In the Address field, enter one of the following URLs:
   - For WebSphere: **http://*domain.name*:9080/*appName*/index.html**
   - For WebSphere Express: http://***domain.name***:7080/***appName***/index.html
   - For WebLogic: **http://*domain.name*:7001/*appName***

   where:
   - ***domain.name*** is the fully qualified domain name of the application server.
   - ***appName*** is the name of the application you want to check on.

   The Web browser displays the **index.jsp** page for the designated application.

Once the **index.jsp** page is open, there are four options, described in the following sections:

- Validating the Application Deployment
- Viewing the Application in HTML Mode
- Monitoring Application Performance
- Viewing Application Trace and Report Logs

Validating the Application Deployment

The utility for validating application deployments is designed to help ensure that the application has been properly deployed on the application server. While it is not guaranteed to catch all deployment errors, it usually catches the vast majority of them.

To use this utility:

1. Access the **index.html** page.

   To access this page, see Accessing the Application Deployment Utilities.

2. Click **Validate** (in the bullet **Validate the application dependencies**).

   The Web browser displays the validation page for the selected application.

3. Inspect the validation page for any sign of a problem with the installation and deployment of the application on the application server.

   Tip:
   In general, green is good. Green text indicates that a setting is properly configured. Yellow text indicates that an optional setting is not as expected, but in most cases, it does not indicate a condition that would prevent the application from running properly. Red indicates that a required setting is not right and, therefore, that the application probably will not function as expected.

This page is divided into three basic areas:

1. **Application details** - Provides information about:
   - Dialog Designer version, framework runtime version (scert.jar), framework runtime common version (scertcommon.jar), Axis version (axis.jar), starting language, platform, ASR, SSML usage, and WebLM URL.
   - License status for platform, including whether runtime, CTI and IC connectors are enabled, or not.
   - HTTP and HTTPS Proxy settings
2. **General Dependency Area Details** - Provides information for the following dependancy areas:
   - **Libraries** - Displays the status of logging, licensing, and encoding run-time support library files that must be installed for the application to run properly.
   - **Web Services** - Displays the status of client Web services that must be installed for the application to run properly. The following Web services are searched for using the Validate option:
     - **org.apache.axis.Version** found in **<*tomcat dir*>\common\lib\axis.jar**
     - **javax.xml.rpc.Service** found in **<*tomcat dir*>\common\lib\jaxrpc.jar**

  - **org.apache.commons.discovery.ResourceDiscover** found in **<*tomcat dir*>\common\lib\commons-discovery-0.2.jar**
  - **org.apache.commons.logging.LogFactory** found in **<*tomcat dir*>\bin\commons-logging-api.jar**
  - **javax.activation.DataSource** found in **<*tomcat dir*>\common\lib\activation.jar**
  - **javax.xml.soap.SOAPBody** found in **<*tomcat dir*>\common\lib\common\lib\saaj.jar**
  - **javax.wsdl.PortType** found in **<*tomcat dir*>\common\lib\wsdl4j-1.5.1.jar**
- **Database** - Displays the status of database connectivity files that must be installed for the application to run properly, including:
  - Jar files used by Tomcat for connection pooling
  - Installed JDBC driver files (application server dependent)
  - Testing option to validate data sources added in Dialog Designer
- **Interaction Center** - Displays the status of the Interaction Center connector which is relevant only if an application will be using the IC connector.
- **CTI (Computer Telephony)** - Displays the status of the CTI connector which is relevant only if an application will be using the CTI connector.

3. **Required dependencies** - Displays information about any other modules that the application requires in order to run properly.

   Any modules that an application depends on must reside on the same application server. If the module is missing, the validation utility flags it as an error. If the version is different or the application and module are set to use different versions of the run-time support files, the validation utility flags a potential error.

Note:

If you are storing "custom object" (objects that you define) in session properties and wish to pass them between to modules (Web applications), the code for the objects must be in the **<tomcat_home>/common/lib** so that the class loader works properly.

If that code is on **<appname>/WEB-INF/lib** then an object instantiated in project A cannot be accessed on project B. By moving the implementation of the class to the common lib and in turn the common class loader, the implementation may be shared across Web applications.

See Preparing the Application Server to Run Dialog Designer Applications.

Viewing the Application in HTML Mode

The utility to view an application in HTML mode is provided as a debugging tool. This utility allows you to step through an application that has been deployed, checking it at each step to make sure that prompts and actions are working as expected. It does not require that you make a call to the system, but rather translates the application's activity into an HTML mode that allows you to interact with the application in a simulation mode through a Web browser.

To view an application in HTML mode:

1. Access the **index.html** page.

   To access this page, see Accessing the Application Deployment Utilities.

2. Click **View** (in the bullet **View the application in HTML mode**).

   The Web browser starts the application. The displayed page is split into two parts:

   - The root page for the selected application.

     The same root page information is displayed on every page of the application.

   - A **Start application** settings section that allows you to preset and simulate various options that an application may expect to use during its run.

     For example, if your application is designed to make use of the DNIS, you must supply the DNIS you want to simulate in the **DNIS** field.

   At the bottom of each page are the following buttons.

Button	Function/Comments
Continue	Moves the application to the next page.
Show Doc VXML	Displays, in a separate browser window, the generated VoiceXML code for the current page.
Show Root VXML	Displays, in a separate browser window, the generated VoiceXML code for the root page.

Note: The **Show ... VXML** buttons are intended primarily as debugging aids. Clicking these buttons may not work as expected in every case, because they require the Web browser to send another request for the VXML contents. This request can cause session variables to change and alter the test flow of the application.

Note: In a Firefox Web browser, only the system outputs (prompts) are displayed. To see the VoiceXML code, use Internet Explorer.

Monitoring Application Performance

The performance monitoring utility allows you to view and analyze a variety of statistical data associated with system performance while running the application.

To use this utility:

1. Access the **index.html** page.

   To access this page, see Accessing the Application Deployment Utilities.

2. Click **Performance Monitoring** (in the bullet **Performance Monitoring for the application**).

   The Web browser displays the performance monitoring page for the selected application.

This page is divided into three basic areas:

- **Application details** - Displays the name and version of the application.
- **Performance statistics** - Displays statistics related to the application's performance in five columns:
  - **Counter** - Names the operation or method in the application for which statistical data is being displayed on that row.
  - **Average (ms)** - Displays the average amount of response time for all calls to the designated operation or method, in milliseconds.
  - **Min (ms)** - Displays the fastest response time among all calls to the designated operation or method, in milliseconds. A negative one (**-1**) in this column indicates that no performance data has been captured.
  - **Max (ms)** - Displays the slowest response time among all calls to the designated operation or method, in milliseconds. The text, if any, displayed in square brackets [ ] is the name of the operation or method.
  - **Total Samples** - Displays the total number of times the designated operation or method was called during the current sampling period.
- **Command buttons** - At the bottom of the page are three command buttons:
  - **Enable/Disable** - Toggles between **Enable** and **Disable**, according to its current state. When performance monitoring is disabled, this button is labeled **Enable**, and vice versa.

    Before data can be collected for analysis, performance monitoring must be enabled for the application. To turn off performance monitoring for the application, click **Disable**.

Note:
Enabling performance monitoring causes the system to add statistical data to a file every time the application runs. If performance monitoring is left on all the time, this file can eventually become enormous in size. Therefore, it is recommended to only use performance monitoring when needed for debugging or statistical analysis.

- **Refresh** - Use this button when you have two browser windows open. For example, when using one browser window to view the application in HTML mode and a second window to display the performance statistics. Using this button refreshes the data after running several tests of the application.
- **Reset** - This button clears all statistical data in the performance monitoring data file and resets all values to zero (for times) or null (for method or operation names).

  Use this button when you want to flush the results of previous tests and start over.

Viewing Application Trace and Report Logs

The application trace and report logs option displays a variety of statistical data associated with system performance while running the application.

To use this utility:

1. Access the **index.html** page.

   To access this page, see Accessing the Application Deployment Utilities.
2. Click **Application Trace and Report Logs**.

   This utility allows the viewing of three common log files:

   - **report.log** - Contains data related to the use of report items in the call flow.
   - **savereport.log** - For Avaya Voice Portal applications only, this log contains report data that has been queued to the VPMS, but for some reason the system has not been able to send the data as scheduled.
   - **trace.log** - Contains the VXML generated when running the application, as well as any trace item data that may have been generated within the call flow.

   For information on enabling and disabling these logs, see Trace.

Appendix A: Nodes and Palette Options

Avaya Dialog Designer uses nodes in the Call Flow Editor as the basic building blocks for speech application projects. Each node represents a piece of code and a functionality. The Call Flow Editor palette provides access to the nodes and action options you use to build call flows. For more information about the basic types of nodes, see Getting Familiar with Nodes to Build Applications.

Palettes are the lists, or menus, that appear on the left side of most editors in Dialog Designer. Palette options are the options and node items that appear in one or more palettes of the editors in Avaya Dialog Designer.

This appendix provides detailed information about the nodes, options, and items that appear on the various palettes in Dialog Designer editors. It contains the following sections:

- Detailed Node Descriptions
- Detailed Palette Option Descriptions

Detailed Node Descriptions

The Call Flow Editor uses a palette to provide access to the nodes you use to build call flows in Dialog Designer speech projects. Dialog Designer offers three types of nodes to build projects. that are available from the Call Flow Editor palette. They are grouped depending on whether you have imported any module nodes into the Dialog Designer workbench.

The following detailed information is provided for each of the nodes in Dialog Designer:

- **Type** - Describes what type of node it is. For more information about the three types of nodes, see Getting Familiar with Nodes to Build Applications.
- **Purpose** - Describes what the node is designed to do.
- **Behavior** - Describes in detail how the node works, including interdependencies with other nodes, options, or items.
- **Properties** - Indicates what properties, or attributes, are associated with the option or item in the **Avaya Properties** view. Describes in detail valid values.
- **Default node structure** - Indicates what node options or items the node is populated with (in the node detail editor) when you select it and place it in the call flow.

Note:
You can change the default flow in the node editor any way you want, but be careful; nodes can be inadvertently broken so that it no longer performs the function it was intended to perform.

Following is a quick link list to available nodes:

Announce Node 

Type

Template (composite) node

Purpose

The purpose of this node is to play an announcement to the caller.

Behavior

When you place this node in a call flow, you must define a prompt to play to the caller. This node, then, plays that prompt when the node is executed.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.

- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

  Tip:
  Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

- **Prompt** item - Indicates what prompt Dialog Designer must use to play the announcement.
- **Next** item - Indicates what node comes next in the call flow.

Blind Transfer Node 

Type

Template (composite) node

Purpose

The purpose of this node is to transfer a caller without verifying that the transfer can be successfully completed or retaining control of the call. In a blind transfer, the system dials the number and then transfers the caller without any further checking or other action. Once the caller is transferred, the application continues to execute until finished, and the caller does not return to the application.

Note:
Support for call transfer functions can vary from one IVR system to another. For more information and details about support for transfer functions on your system, see the documentation for your IVR system.

Behavior

When you place this node in the workspace, Dialog Designer automatically creates a variable with the same name. This is because the default flow in the node editor for this node contains a Blind Transfer item. For more information about this item and the associated variable, see Blind Transfer.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.

- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

  Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

- **Blind Transfer** item - Indicates the telephone number, either hard-coded or contained within a variable, to which the application transfers the caller. Includes an option to play a prompt (**Prompt** item) to the caller before performing the transfer.
- **Next** item - Indicates what node comes next in the call flow.

Bridged Transfer Node 

Type

Template (composite) node

Purpose

The purpose of this node is to transfer a caller after verifying that the transfer can be successfully completed, while retaining control of the call. In a bridged transfer, the system dials the number, waits for a response from the dialed number, and then transfers the caller. If the dialed number is busy or there is no answer, the system responds in a prescribed manner, without losing control of the call. If the transfer is successful, the application remains in a suspended mode until the transfer call is ended. At that point, control returns to the application, which continues executing. For more information on bridged transfers, see Bridged Transfer.

Note:
Support for call transfer functions can vary from one IVR system to another. For more information and details about support for transfer functions on your system, see the documentation for your IVR system.

Behavior

When you place this node in the workspace, Dialog Designer automatically creates a variable with the same name. This is because the default flow in the node editor for this node contains a Bridged Transfer item. For more information about this item and the associated variable, see Bridged Transfer.

Note:
Any audio prompts in, or after a transfer node, will not get played before the call has been transferred. Putting the audio prompt as a transitional prompt when transitioning to the node that does the transfer ensures that the prompt is played before the transfer happens.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Tip:
Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

- Bridged Transfer item - Indicates the telephone number, either hard-coded or contained within a variable, to which the application attempts to transfer the caller. This item includes:
  - An option to play a prompt (Prompt item) to the caller before performing the transfer.
  - The capability to have the application respond to a specific word, phrase, or touchtone key press (Grammar item). This might be useful, for instance, if you wanted to allow callers to return to the main application by making two consecutive pound (#) key presses or uttering a phrase such as “Avaya come back.”
- Next item - Indicates what node comes next in the call flow.

When you place this node in the workspace, Dialog Designer automatically creates a variable with the same name.

Consultation Transfer Node

Type

Template (composite) node

Purpose

The purpose of this node is to transfer a caller while retaining control of the call if the transfer attempt is unsuccessful. In a consultation transfer, the system dials the number and monitors the progress of the transfer.

When the transfer has successfully completed, the caller is disconnected from the application (the caller is now connected with the called party) and the application will continue to execute until finished. If the transfer attempt is unsuccessful, the call is brought back to the application and the application may continue to interact with the caller (playing prompts, collecting inputs, or attempting other transfers, etc.).

Note:
Support for call transfer functions can vary from one IVR system to another. For more information and details about support for transfer functions on your system, see the documentation for your IVR system.

Behavior

When you place this node in the workspace, Dialog Designer automatically creates a variable with the same name. This is because the default flow in the node editor for this node contains a Consultation Transfer item. For more information about this item and the associated variable, see Consultation Transfer.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

  Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.
- **Location** - (Read-only. May be disabled in Call Flow Preferences) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

- Consultation Transfer item - Indicates the telephone number, either hard-coded or contained within a variable, to which the application transfers the caller. Includes an option to play a prompt (Prompt item) to the caller before performing the transfer.
- Next item - Indicates what node comes next in the call flow.

Data Node

Type

Application Item (basic) node

Purpose

Use this node to perform a variety of operations on data obtained from sources internal or external to the application or the system on which the application is running. This node is intended primarily to make it possible to:

- Manipulate application or system data.
- Make use of Web services or database operations.
- Perform conditional branching based on variable data.
- For IC connectors:
  - Get VDU Fields - Gets multiple VDU values (up to 20) in one call
  - Set VDU Fields - Sets multiple VDU values (up to 20) in one call.
- For CTI connectors:
  - Blind Call (CTI) - Does a hold/dial/transfer in one command. The call is transferred once placed so you do not know the status of the call.
  - Consultation Call (CTI) -Does a hold/dial/transfer in one command. The status of the call is reported back to the application. If the call fails, it is retrieved before returning back to the application.

Behavior

When placed in the call flow, this node sets up a generic form in which the palette is optimized for the purposes described in the previous section. In addition, this is the node in which you can set up and use functions with Interaction Center (IC) and Computer Telephony Integration (CTI) connectors. For more information about IC and CTI connectors and their use, see CTI and IC Connectors.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Tip:
Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

- **Next** item - Indicates what node comes next in the call flow.

Disconnect Node 

Type

Template (composite) node

Purpose

Use this node when you want to disconnect the caller from the system or a VXML session. This node is usually used when you must perform some type of “clean up” or follow-up actions. It is also used to return data back to other interpreters (for example, a CCXML processor).

Behavior

When you place this node in the workspace, the application disconnects the caller and proceeds to the next node, as indicated in the On Disconnect item.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Tip:
Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

- **On Disconnect** item - Indicates what node the application goes to after disconnecting the caller.

Form Node 

Type

Application Item (basic) node

Purpose

Use this multipurpose node to perform a wide variety of functions, from collecting and processing caller input to initiating call transfers and more. This node is the most generic of all the nodes and is actually the basis of many of the Templates (Composite Nodes).

Behavior

When placed in the call flow, this node sets up a generic form that you can use for a wide variety of purposes. In terms of VoiceXML, this node sets up a generic form that meets the broad general guidelines for a form as defined in the W3C VoiceXML 2.0 Recommendation.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

  Tip:
  Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.
- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

There is no default flow in the node editor for this node.

Menu Node 

Type

Application Item (basic) node

Purpose

The purpose of this node is to present the caller with a limited series, or menu, of options. The caller must respond to these options using either touchtone (DTMF) key presses or spoken responses.

Behavior

When placed in the call flow, this node sets up a framework to construct a menu of options for callers, mostly through the options offered on the node editor palette. One palette option, in particular, is optimized for use with this node: the Choice item. Use this item, along with Prompt and Grammar items, to designate the options available to the caller, thus creating the menu.

When you place an **Menu** item in the call flow, Dialog Designer automatically creates a complex variable with the same name. A complex variable contains the following fields, which are populated upon completion of the node:

- **confidence** - A number between 0.0 and 1.0 that expresses the degree of confidence the system has that it has correctly recognized the caller utterance. The number 0.0 indicates minimum confidence, and the number 1.0 indicates maximum confidence. The specific interpretation of this number depends on the ASR platform.
- **utterance** - A text (raw string) summary of what the caller said or keyed in, as recognized by the ASR engine. In the case of a DTMF response, this is the digit sequence of keys that the caller pressed.
- **inputmode** - The mode in which the caller input was provided, either voice or DTMF.
- **interpretation** - The contents of the interpretation tag, as provided by the ASR engine. In other words, this is the result of how the ASR engine recognized and mapped the utterance. If the ASR engine is unable to provide an interpretation, this field remains undefined.
- **value** - If the ASR engine returned an interpretation, this field takes on the same value as the **interpretation** field. If not, this field takes on the value of the **utterance** field (or other optional fields depending on the **Record Utterance** and **Collect Mark Data** properties).

Once the ASR engine returns a recognition result and populates these Input variable fields, you can use the outcome to further direct the call.

Tip:

It is always a good idea to give callers the option to use either a spoken reply or a DTMF key press to select a choice. For example, you can create a prompt for a choice that says, “To hear about your savings options, press one or say "savings." Then, when setting up the Choice item, you both use an ASR grammar for “savings” and enter **1** in the DTMF field.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Tip:

Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Record Utterance** - (False/True). Used for capturing the caller’s spoken utterance.
- **Collect Mark Data** - (False/True). Used for capturing the most recently encountered mark in a prompt.
- **Record Utterance** – (false/true). When true, additional fields are added to the variable that store the recording of the caller's utterance, the size of the recording (in bytes) and the duration of the recording (in milliseconds).
- **Collect Mark Data** – (false/true). When true, additional fields are added to the variable that store the name of the last Mark encountered by the VoiceXML platform as well as the time elapsed since the mark was processed.
- **Location** - (Read-only) Displays the location of the node graphic in the workspace.
  - **Bind Name to Node** – (true/false). When true, the name of the variable item is bound to the name of the node. When the parent node is renamed, the name of the variable item is renamed as well; keeping the name of the variable in sync with the node name.

Default Node Structure (Node Detail Editor Flow)

There is no default flow in the node editor for this node.

Prompt and Collect Node 

Type

Template (composite) node

Purpose

The purpose of this node is to prompt a caller for a response and collect that response in the form of a spoken reply or DTMF (touchtone) key presses.

Behavior

When you place this node in the workspace, Dialog Designer automatically creates a variable with the same name. This variable contains the following fields:

Tip:

The returns from all these variable fields can be used or manipulated later in the application. For example, if you wanted to ascertain the range of recognized utterances as received by the ASR server, you might have the contents of the utterance field written to a database. You can use this information later to refine and improve the functioning of your application.

- **confidence** - A number in the range 0.0 to 1.0 that indicates the degree to which the ASR server is confident that it has correctly recognized the spoken response. 0.0 indicates minimum confidence and 1.0 indicates maximum confidence.
- **utterance** - The raw string of words that the ASR server recognized. The presentation and spelling of these words is platform specific. For example, the ASR server might present the number 530 as “five hundred thirty,” “5 hundred 30,” or even “530.” If DTMF key presses were used, this field contains the string of matched digits.
- **inputmode** - The mode in which the caller input was provided, either voice or DTMF.
- **interpretation** - The interpretation, by the ASR server, of the spoken response. This interpretation is determined by the programming of the ASR server itself. This field also contains the tag of the grammar item that the ASR server recognized. The ASR server must be set to interpret the response, including the tag, correctly.

Tip:

When you want to use the spoken response of a caller to control the flow of the call, use the **interpretation** field, rather than the **utterance** field. Avaya recommends that you use the **interpretation** field in this type of conditional branching situation, because the tag of the grammar item is not language dependent. This fact means that it does not matter whether the actual **utterance** of the caller was “yes,” “oui,” “ja,” or some other language form. If the grammar tag was set to interpret each of these responses as “true,” the **interpretation** field maps the response to a value of “true” in each of these cases. Thus, this tag mapping helps the application to use the response correctly, no matter what form the caller uttered.

- **value** - If the ASR server returns a value for the **interpretation** field, this field contains the same value. If the ASR server does not return a value for the **interpretation** field, this field contains the value of the **utterance** field (or other optional fields depending on the **Record Utterance** and **Collect Mark Data** properties).

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Tip:

Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Record Utterance** – (false/true). When true, additional fields are added to the variable that store the recording of the caller's utterance, the size of the recording (in bytes) and the duration of the recording (in milliseconds).
- **Collect Mark Data** – (false/true). When true, additional fields are added to the variable that store the name of the last Mark encountered by the VoiceXML platform as well as the time elapsed since the mark was processed.
- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

- Input item - Uses sub-items to indicate:
  - What prompt the call flow uses to collect input from the caller (Prompt item).
  - What means the call flow uses to collect the input (ASR or DTMF Grammar item).

  - How the system functions if the caller does not respond (No Input item, which has its own associated Prompt item).

    Note that this setting overrides any global, or application-level, settings you might have assigned in the **AppRoot** node. If you prefer to use the global settings, you can delete this item. For more information about global settings, see AppRoot Node Event Handlers.

  - How the system functions if the system cannot match the response by the caller with any expected response (No Match item, which has its own associated Prompt item).

    Note that this setting overrides any global, or application-level, settings you might have assigned in the **AppRoot** node. If you prefer to use the global settings, you can delete this item. For more information about global settings, see AppRoot Node Event Handlers.

- Next item - Indicates what node comes next in the call flow.

Record Node 

Type

Template (composite) node

Purpose

The purpose of this node is to prompt the caller for a spoken reply and then to record the response by the caller. The node then saves the recording in the **data/temp** folder.

Behavior

When you place this node in the workspace, Dialog Designer automatically creates a variable with the same name. This variable contains the following fields, which are populated on completion of the node:

- **duration** - The length of the recording in milliseconds.
- **size** - The size of the recording in bytes.
- **termchar** - The value of the DTMF key the caller pressed to terminate the recording. If the caller did not press a DTMF key to terminate the recording, this field is undefined.
- **maxtime** - Boolean value that indicates whether the recording was terminated because the caller exceeded the maximum allowable time for the recording (**true**) or not (**false**).
- **utterance** - A text summary of what the caller said or keyed in to terminate the recording.
- **confidence** - A number in the range 0.0 to 1.0 that indicates the degree to which the ASR server is confident that it has correctly recognized the termination key word or phrase. 0.0 indicates minimum confidence and 1.0 indicates maximum confidence.

- **value** - The URL to the audio (*.wav) file that contains the recording of the response of the caller.

Tip:

The return values in these variable fields can be used or manipulated later in the application. For example, if you want to play the recording back for the caller to approve, you can create a prompt that uses a phrase variable with a format of **url**. You can then assign the **Record** variable's **value** field to that phrase variable. Or, if you want to assess the general confidence levels of the ASR server, you can record the return from the **confidence** field to a database for later report processing and analysis.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Tip:

Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

- Record item - Use this to set options for recording a spoken response. This item uses sub-items to indicate:
  - What prompt the application uses to obtain a spoken response from the caller (Prompt item).
  - What node the application uses to handle the recording and finish the call transaction if the caller ends the recording by hanging up (On Disconnect item).
  - How the system functions if the caller does not respond (No Input item, which has its own associated Prompt item).

    Note that this setting overrides any global, or application-level, settings you might have assigned in the AppRoot node. If you prefer to use the global settings, you can delete this item. For more information about global settings, see AppRoot Node Event Handlers.
- Next item - Indicates what node comes next in the call flow.

Return Node 

Type

Application Item (basic) node

For sub flow returns, see Sub Return Node.

Purpose

Use this node to exit the application. If you are using the speech project as a module for another application, this node can return the focus and pass parameters to the main application project or to yet another module.

Behavior

The behavior of this node depends on whether the speech project is intended for use as a standalone or parent application or as a reusable module.

In a standalone or parent application, this node tells the VoiceXML browser to terminate the application.

In a reusable module, this node tells the VoiceXML to return to the parent application or to proceed to another module. Either way, this node also makes it possible to define output parameters that are passed to the receiving application or module.

For either use, this should be the last node in your call flow. If the speech project is a standalone or parent application, this node is not required, and the application should still successfully terminate. However, Avaya recommends that you include this node anyway. If the speech project is a reusable module, you *must* have this node as the final node in the module.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

  Tip:
  Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.
- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

There is no default flow in the node editor for this node.

Servlet Node

Type

Application Item (basic) node

Purpose

Use this node to insert custom Java code into your project. You can insert, for example, code written for some business logic or code that calls other Java code, such as a Java bean, that Dialog Designer does not natively support.

Behavior

When you place this node in a call flow, Dialog Designer does nothing to process it. During code generation, however, Dialog Designer generates the appropriate Java classes in the WEB-INF directory to contain and run the code “as is.”

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

  Tip:
  Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.
- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

There is no default flow in the node editor for this node.

Sub Begin Node

Type

Application Item (sub flow) node

Purpose

Sub flows are used for invoking another callflow in the project. Large projects may be broken up into multiple, smaller callflow files which are invoked by other project callflows using the Sub Flow node.

The Sub Begin Node is the entry point for a sub call flow. Somewhat like the AppRoot/Start node for the main call flow, but the Begin node cannot be edited and does not allow items to be added to it like the AppRoot.

There are two other parts/nodes to a sub flow multiple call flow feature:

- Sub Flow Reference Node – This defines a reference to a sub flow that has been defined in the project (). A sub flow can be referenced multiple times within a call flow.
- Sub Return Node – This is the return point from a sub flow, returning control back to the call flow (called the sub flow). A sub flow can have multiple return nodes. This is similar to the Return Node from the main call flow, but the sub flow return does not allow items (output parameters, etc.) to be added to it like the main flow Return node does.

Behavior

Contents of sub flows are hidden to other flows (i.e., “black boxes”). You cannot use a “goto” to go to other nodes in other flows.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Flow Name** - Name of the main flow that the Sub flow is associated with.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Default Node Structure (Node Detail Editor Flow)

There is no default flow in the node editor for this node.

Sub Flow Reference Node 

Type

Application Item (sub flow) node

Purpose

Sub flows are used for invoking another callflow in the project. Large projects may be broken up into multiple, smaller callflow files which are invoked by other project callflows using the Sub Flow node.

The Sub Flow Reference Node defines a reference to a sub flow that has been defined in the project. A sub flow can be referenced multiple times within a sub flow.

There are two other parts/nodes to a sub flow multiple call flow feature:

- Sub Begin Node – This is the entry point for a sub call flow. Somewhat like the AppRoot/ Start node for the main call flow, but the Begin node cannot be edited and does not allow items to be added to it like the AppRoot.
- Sub Return Node – This is the return point from a sub flow, returning control back to the call flow (called the sub flow). A sub flow can have multiple return nodes. This is similar to the Return Node from the main call flow, but the sub flow return does not allow items (output parameters, etc.) to be added to it like the main flow Return node does.

Behavior

Contents of sub flows are hidden to other flows (i.e., “black boxes”). You cannot use a “goto” to go to other nodes in other flows.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Flow Name** - Name of the main flow that the Sub flow is associated with.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Default Node Structure (Node Detail Editor Flow)

There is no default flow in the node editor for this node.

Sub Return Node

Type

Application Item (sub flow) node

Purpose

Sub flows are used for invoking another callflow in the project. Large projects may be broken up into multiple, smaller callflow files which are invoked by other project callflows using the Sub Flow node.

The Sub Return Node is the return point from a sub flow, returning control back to the call flow (called the sub flow). This is similar to the Return Node from the main call flow, but the sub flow return does not allow items (output parameters, etc.) to be added to it like the main flow Return node does.

There are two other parts/nodes to a sub flow multiple call flow feature:

- Sub Begin Node – This is the entry point for a sub call flow. Somewhat like the AppRoot/ Start node for the main call flow, but the Begin node cannot be edited and does not allow items to be added to it like the AppRoot.
- Sub Flow Reference Node – This defines a reference to a sub flow that has been defined in the project (). A sub flow can be referenced multiple times within a call flow.

Behavior

Contents of sub flows are hidden to other flows (i.e., “black boxes”). You cannot use a “goto” to go to other nodes in other flows.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Flow Name** - Name of the main flow that the Sub flow is associated with.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Default Node Structure (Node Detail Editor Flow)

There is no default flow in the node editor for this node.

Symbolic Node 

Type

Application Item (basic) node

Purpose

Use this node as a “goto” node, to redirect the application to another node. This node is useful when you want several nodes all to go to the same destination node or just to keep from cluttering up the workspace with a confusing web of connector lines.

Behavior

When you place a Symbolic node in the Call Flow Editor main workspace, you must set its one property to go to some other node during simulation and runtime, then the application automatically jumps to the indicated node.

When you set the node for the Symbolic node to go to, Dialog Designer automatically places a Symbolic node icon element ( ) in the upper left area of the node icon to which the Symbolic node goes.

When you click a Symbolic node in the main workspace, Dialog Designer automatically highlights in yellow the node that the Symbolic node is set to go to. This highlighting makes it easier to locate the destination of the Symbolic node.

Further, Symbolic node references can quickly duplicated from a selected node in the Call Flow Editor by right-clicking on any existing node, and selecting **Create New Reference** from within the context menu. A

Properties

Because the Symbolic node is simply a pointer to another node, its properties, as displayed and set in the **Avaya Properties** view, are different from all the other nodes:

- **Location** - (Read-only) Displays the location of the node graphic in the workspace.
- **Node** - Indicates the node that the Symbolic node points to. You can use the drop-down list to select the node you want to point to.

Default Node Structure (Node Detail Editor Flow)

There is no default flow in the node editor for this node.

Tracking Node 

Type

Application Item (basic) node

Purpose

Use this node to log:

- Debug and tracing information about the application.
- Event and alarm data to a system log.
- Performance and other data that can be put into a report.

Behavior

In Voice Portal (VP) systems, error and report data are sent to a log file on the Voice Portal Management System (VPMS), from which you can generate reports to analyze application operations.

The VP system logs the trace/debug data locally on the application server. The path to the log file is:

***/applicationName*/data/log/trace-*YYYY-MM-DD*.log**

where:

- ***applicationName*** is the name of your application project

- ***YYYY-MM-DD*** is the year-month-day that the log file was generated

On Avaya IR systems, all logs are local to the application server. The Dialog Designer application places them in the ***//applicationName/*data/log** directory. You can view the data and generate reports from the log files located there.

The trace for the current day is ***//***applicationName/data/log/trace.log**.

Note:
Open and review logs using a text editor, or import the data into a report generating application. Except for the Application Report on the Avaya Voice Portal system, Avaya does not provide reporting software in the current release.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Tip:
Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

- **Next** item - Indicates what node comes next in the call flow.

VXML Servlet Node

Type

Application Item (basic) node

Purpose

Use this node to insert custom VoiceXML code into your project. The purpose of this node is to make it possible to write and employ custom VoiceXML code for operations that Dialog Designer does not natively handle.

Behavior

The framework for this node generates the VoiceXML header and footer (<vxml> and </vxml>) and the form header and footer (<form> and </form>). The custom code you place within the Java class for this node is inserted between the form header and footer.

After you place this node in a call flow, you must edit the Java class for the node. Because this node permits you to write and employ your own VoiceXML, Avaya can assume no responsibility for the results. It is possible to write code that will prematurely exit the application, transfer a caller out of the system, and perform other actions that the parent application is then not aware of. This can cause hung applications, run-time errors, and other problems. Avaya recommends that you use this node with caution.

To edit the Java class for the node:

1. Place a VXML Servlet node in the call flow.
2. Designate the next node for the VXML Servlet node to proceed to when finished.
3. To save the project, in the Navigator view, select the project and then press **Control+Shift+S**.

   Note:
   Before editing the Java class, connect the VXML Servlet node to whatever node is next in the call flow and save the speech project. Dialog Designer does not allow editing of the Java class until steps 2 and 3 are completed.
4. After the project is finished building, right-click the VXML Servlet node and, from the pop-up context menu, select **Edit *VXMLservlet*.java**, where *VXMLservlet* is the name of your VXML servlet node.
5. In the Java class editor, add the method:

   public void markupLanguageGeneration(PrintStream out, Submit submit, SCESession session) {

   and add to this method your custom VoiceXML code.

   Note:
   The ability to integrate variables into and from this node is not easily supported. Avaya recommends that, if possible, you avoid using this node to pass variable values.

Properties

- **Name** - Enter a name that reflects the central purpose of the node. Use naming conventions for Java components. See Naming Java Components.
- **Comments** - Enter anything you want to add as a reminder or description of the purpose or content of the node. You can leave this field blank.

Tip:

Dialog Designer uses whatever text you enter in this property field for the pop-up hint that appears when your mouse hovers over the node icon in the workspace. If you leave this blank, Dialog Designer uses the name of the node for the pop-up hint.

- **Location** - (Read-only) Displays the location of the node graphic in the workspace.

Default Node Structure (Node Detail Editor Flow)

- **Next** item - Indicates what node comes next in the call flow.

Detailed Palette Option Descriptions

Most of the editors in Dialog Designer use palettes as a way of providing the options and node items that can be used within each editor.

The following information is provided for each of the options and items in this section:

- **Type** - Identifies which type of option or item it is. This helps identify what the option or item is used for.
- **Available from** - Indicates where in Dialog Designer the option or item can be used.
- **Purpose** - Describes what the option or item is designed to do.
- **Behavior** - Describes in detail how the option or item works, including interdependencies with other options or items.
- **Properties** - Indicates what properties, or attributes, are associated with the option or item in the **Avaya Properties** view. Describes in detail valid values.

Alarm

Type

IC item

Available from

Data Node only

Purpose

The **Alarm** item makes it possible to send an alarm from the speech application to the Alarm Manager on the Interaction Center (IC) system.

Behavior

Before you can use this or any other IC item, you must enable IC for Dialog Designer. For more information about the ability of Dialog Designer to interact with IC, and to enable IC in your Dialog Designer applications, see About the IC Connector.

When the speech application encounters the **Alarm** item, it passes the values for the properties to the IC VOX server, which relays them to the IC system.

Properties

- **Name** - Enter the name to assign the alarm.

  This name is passed along to the IC system, where the name is used as an identifier.

- **Priority** - Select the priority to assign the alarm. Options include:
  - **Low**
  - **High**
  - **Info**
  - **EMERGENCY**

  The priority you assign here determines the alarm that is generated on the IC system. For the interpretation of what these alarms mean, see your IC documentation.

- **Description** - Enter in this field a short description of what the alarm is for. This description is passed verbatim to the Alarm Manager on the IC system, as part of the alarm event.

Audio Variable 

Type

Prompt Editor Palette item

Available from

Any level tab in the Prompt File Editor. For more information about the Prompt File Editor, see Using the Prompt File Editor.

Purpose

Audio variables make use of prepackaged and prerecorded variables to play back standardized types of information to callers. These standardized types of information include time, date, and currency, to name a few. Audio variables make it possible to use prerecorded speech, rather than Text-to-Speech, to play this type of dynamically changing information to callers.

Behavior

The runtime framework treats audio variables in speech applications differently than other types of variables. When runtime encounters an audio variable, it takes the value assigned to that variable and parses it into pieces that correspond with predefined elements. The elements are then mapped to the appropriate pre-recorded (*.wav) files and at run time plays back the prerecorded files.

For example, you might have a variable defined to take the value of the current date. If you then use this variable as an audio variable segment in a prompt, the localization bundle parses the value from the variable. It assigns one part of the variable to “month,” another part to “day,” and yet another to “year.” It then maps each of these parts to the appropriate prerecorded audio files.

Before using audio variables, a localization bundle and standard phrases for the language needs to be installed. To install the localization bundle, see Administering Localization Bundles. To install a set of standard phrases, see Using a Localization Bundle’s Standard Phrasesets.

For audio variables to work correctly, you must be careful that:

- The format of the variable to be used as the audio variable matches the format that the localization bundle expects and can recognize. For more information about the formats that localization bundles can recognize, see Variable Formats in Localization Bundles.
- You select the correct format for the audio variable when you use it in a prompt. For more information about formats for audio variables, see System Variable Fields and Properties.

For information about how to use system variables as audio variables, see Audio Field Properties.

Properties

- **Variable** - Select the variable you want to use. If the variable you select is a complex variable, you must also select a **Variable Field**.
- **Variable Field** - Select the variable field you want to use.

  The contents of this list depend on which variable you selected in the **Variable** property: Only the fields associated with the selected variable are presented in this list.

  Note:
  If the variable you selected in the **Variable** field is a simple variable, this field is inactive.

- **Format** - Select the format you want to use for the variable.

  The options available depend on the language and the localization bundle you are using. For more information about formats available with localization bundles, see Variable Formats in Localization Bundles.

Blind Transfer 

Type

Form item

Available from

- Announce Node
- Blind Transfer Node, pre-populated by default
- Form Node

Note:
Although the **Blind Transfer** item appears on palettes for other nodes, such as the Prompt and Collect Node, the Bridged Transfer Node, and the Record Node, it is not actually available for use in those nodes without destroying their intended functionality. It is intended primarily for use only in the Bridged Transfer Node.

Purpose

This item provides the elements you need to set up a blind transfer. In a blind transfer, a speech application transfers a caller to another number and exits the application without verifying that the transfer was successful. For more information about blind transfers, see Blind Transfer Node.

Behavior

The **Blind Transfer** item takes the value of one of the **Destination**... properties as the telephone number to which to transfer the caller. When this item executes, the system dials the telephone number and then transfers control of the call to the dialed number. After the call is transferred, the application finishes and exits.

When you place this item in a node subflow, Dialog Designer automatically creates a variable by the same name. That variable has only one property: **Name**. This variable is undefined until after the transfer is attempted. If the caller hangs up during the call transfer attempt, before the transfer is completed, then this variable is assigned the value **near_end_disconnect**. If for any other reason the transfer is not able to complete successfully, this variable is assigned the value **unknown**. In all other cases, it remains undefined.

Note:
Support for call transfer functions can vary from one IVR system to another. This item uses the VoiceXML <transfer> tag. For more information and details about support for this tag and for transfer functions on your system, see the documentation for your IVR system.

Tip:

You can use the return value of this variable later in your application. For example, if you wanted to provide data for a report on the success of transfer attempts, you might write the value of this variable to a database for further processing or manipulation. Or, you might want to use the value of this variable to redirect the call flow, based on what value is returned.

Properties

- **Name** - Enter the name you want to assign to the item and the corresponding variable. If you change the name of this item in the **Properties** view, Dialog Designer automatically changes the name of the variable as well.

  Note:

  This name must follow conventions for naming Java components as described in Naming Java Components.

- **Destination Number** - Enter the telephone number to which the system will transfer callers.

  Use this when you want to “hard code” the number. For example, if you wanted to transfer the caller at this point to an operator, you might enter 00, the number to dial for the system operator.

  If you use this field, you cannot use the **Destination Variable** field.

- **Destination Variable** - Select the variable that contains the number to which the system will transfer callers.

  Use this when you want to allow for dynamic transfers. In dynamic transfers, for example, callers can specify the telephone number to call, and that number is stored in a variable. In other dynamic transfers, the system obtains the destination telephone number from some other source and stores it in a variable.

  If the variable you select here is a complex variable, you must also select a **Destination Variable Field**.

  If you use this field, you cannot use the **Destination Number** field.

- **Destination Variable Field** - If the selected destination variable (see previous item) is a complex variable, you must select a field from the drop-down list. If the destination variable is a simple variable, this option is inactive.

- **AAI Data** - Enter application-to-application literal data that you want to pass from this application to the receiving end of the call transfer. This option passes the raw data just as you enter it in this field.

- **AAI Variable** - Select the variable that contains the application-to-application data you want to pass to the receiving end.

  If the variable you select here is a complex variable, you must also select an **AAI Variable Field**.

- **AAI Variable Field** - If the selected **AAI Variable** is a complex variable, select the field that contains the application-to-application data you want to pass to the receiving end.

  If the **AAI Variable** is a simple variable, this drop-down list is not populated.
- **Bind Name to Node** – (true/false). When true, the name of the variable item is bound to the name of the node. When the parent node is renamed, the name of the variable item is renamed as well; keeping the name of the variable in sync with the node name.

Boolean 

Type

Database Operation Condition Operator

Available from

Database Operation Editor Predicate Tab

Purpose

For joining two simple conditions with the Boolean operator (AND or OR) to create a Compound condition.

Behavior

The item must exist inside the Compound item. It can only be added after the first condition (Simple/Join/Compound) has been created.

Properties

- **Operator** – Choose either AND or OR.

Break 

Type

SSML item

Available from

Any level tab in the Prompt File Editor. For more information about the Prompt File Editor, see Using the Prompt File Editor.

Purpose

Speech synthesis engines that are SSML-compliant use a built-in algorithm to apply breaks, or pauses, in the reading of text. This algorithm is based on the linguistic context. There might be times, however, when you want to override the automatic breaks of the speech synthesis engine. Use the **Break** item to insert breaks or override automatic breaks in speech synthesis.

Note:
The **Break** item and its properties function correctly only with SSML-compliant speech synthesis engines. The Microsoft Speech SDK, which is used by Dialog Designer during application simulation, is *not* an SSML-compliant speech synthesis engine, so any settings you make with this item are ignored. For more information about the SSML standard, see the Speech Synthesis Markup Language (SSML) Version 1.0 W3C Recommendation.

Behavior

The specific behavior of the **Break** item depends on the properties you use.

In general, the **Break** item overrides an automatic break as interpreted and applied by the speech synthesis engine. The **Break** item also inserts a break where the speech synthesis engine might not place one.

For example, if you placed the following text into a TTS prompt segment:

"Here comes a break and now the break is over."

The speech synthesis engine might render this with a very slight pause before the word "and." If you want a longer pause at that point, you can divide the TTS text into two segments:

- "Here comes a break"
- "and now the break is over."

Then you can place a **Break** item between them, either to increase the relative length of the pause with the **Strength** property, or to set an exact length for the pause with the **Time** property. For more detailed information about these properties, see the following section, "Properties."

Properties

- **Strength** - With the **Strength** property, you can change the length of an automatic break relative to the default interpretation of the speech synthesis engine. For example, if the speech synthesis engine places a break at a point in the text where you do not want one, you can override it by using a **Strength** setting of **none**.

  Select one of the following settings:

Strength Setting	Description
none	This setting suppresses any automatic breaks that the speech synthesis engine might apply to the text.
x-weak **weak** **medium** **strong** **x-strong**	These settings represent a range of break options. The weaker the setting, the shorter the break is at the point where the **Break** item is inserted. The stronger the setting the longer the break is. The specific application of these settings varies according to the TTS server.

- **Time** - With the **Time** property, you can specify exactly the amount of time that the break lasts, in milliseconds or seconds. For example, if you set the **Time** property to **200ms**, the system inserts a 200 millisecond (or 0.2 second) pause at that point in the prompt.

  Select one of the following settings:

Time Setting	Description
250ms **500ms** **750ms** **1s** **2s** **3s** **4s** **5s**	This setting applies a pause of the exact duration selected at the point where it is inserted. • **ms** = milliseconds • **s** = seconds

Time Setting	Description
custom	With this setting, you can define the exact length of time that you want the pause to last. When you select this setting, Dialog Designer automatically adds the **Custom Break Time [s or ms]** property to the Property view.
Custom Break Time [s or ms]	This setting is available only when the **custom** setting for **Time** is selected. With it, you can determine exactly how long a pause you want to insert. The acceptable format for this setting is a number followed, with no space, by: • **ms** = milliseconds • **s** = seconds For example, if you want to insert a pause of exactly one and one-fourth seconds, enter in this field **1250ms**. **Note:** You cannot use decimals in this field. If you want to use a fraction of a second as your time, you must enter it as milliseconds (**ms**).

Bridged Transfer 

Type

Form item

Available from

- Announce Node
- Bridged Transfer Node, pre-populated by default
- Form Node

Note:
The **Bridged Transfer** item cannot be used in the same node as the Input item, the Blind Transfer item, or the Record item. So, although the **Bridged Transfer** item appears on palettes for other nodes, such as the Prompt and Collect node, the Blind Transfer node, and the Record node, it is not actually available for use in those nodes without destroying their intended functionality. It is intended primarily for use only in the Bridged Transfer node.

Purpose

This item provides the elements needed to set up a bridged transfer. In a bridged transfer, a speech application transfers a caller to another number but does not lose control of the call. See Bridged Transfer Node.

Behavior

The **Bridged Transfer** item takes the value of one of the **Destination**... properties as the telephone number to which to transfer the caller. When this item executes, the system dials the telephone number and then waits for a response. For a complete description of the possible responses and outcomes, see Properties.

Note:
Support for call transfer functions can vary from one IVR system to another. This item uses the VoiceXML <transfer> tag. For more information and details about support for this tag and for transfer functions on your system, see the documentation for your IVR system.

When you place this item in a node subflow, Dialog Designer automatically creates a variable with the same name. This variable contains the following fields, which are populated upon completion of the node:

- **duration** - The length of time that the transfer call lasted, in milliseconds.
- **inputmode** - The mode in which the caller input was provided to return control of the call to the application, either voice or DTMF.
- **utterance** - A text (raw string) summary of what the caller said or keyed in to terminate the transfer call.
- **value** - The value of this field depends on the result of the attempt to transfer the call. The following table shows the possible values this field can take, along with the actions that cause the value to be assigned.

If:	**Then this variable field is:**
The caller hangs up either before or after the transfer is completed.	Undefined, and the system throws a **connection.disconnect.hangup** event.
The caller uses a voice or DTMF command to end the call transfer, either before or after the transfer is complete.	Assigned a value of **near_end_disconnect**.
The party being called ends the transfer call by hanging up.	Assigned a value of **far_end_disconnect**.
The transfer does not complete because the number being called is busy.	Assigned a value of **busy**.
The transfer does not complete because an intermediate network refuses the call.	Assigned a value of **network_busy**.
The transfer does not complete because the party being called does not answer within the time set by the Bridged Transfer item **Connect Timeout** property (see Properties).	Assigned a value of **noanswer**.

If:	Then this variable field is:
The transfer does not complete, but the reason for the failure is not known; or the transfer completes but is ended for reasons that are not known.	Assigned a value of **unknown**.
The system ends the transfer call because the transfer exceeds the maximum allowable time as set in the Bridged Transfer item **Maximum Time** property (see Properties).	Assigned a value of **maxtime_disconnect**.
The network ends the transfer call by disconnecting the two parties.	Assigned a value of **network_disconnect**.

Tip:

You can use or manipulate the return values from these variable fields later in the application. For example, if you wanted to generate a report that included information about the average length of time call transfers were taking, thus tying up channels, you might set up the application to write the return value of the **duration** field to a database. You can then later extract the information from the database for your report. Or, you might want the system to wait five seconds before attempting to redial the transfer call after getting a busy signal. You can use the **busy** or **network_busy** return in the **value** field to reroute the call back through the Bridged Transfer node after a five-second pause.

Properties

- **Name** - Enter a name to assign to the **Bridged Transfer** item and its corresponding variable.

  If the name of this item is changed in the **Properties** view, Dialog Designer automatically changes the name of the associated variable as well.

  Note:
  This name must follow conventions for naming Java components as described in Naming Java Components.

- **Transfer Audio** - Select the prompt to use for the call transfer.

  The **Transfer Audio** prompt must be a transitional audio prompt. This transitional audio prompt is the audio (*.wav) file that is played to callers while the call is being transferred. For more information about transitional audio prompts, see About Transitional Audio Prompts.

Note:

Any audio prompts in, or after a transfer node, will not get played before the call has been transferred. Putting the audio prompt as a transitional prompt when transitioning to the node that does the transfer ensures that the prompt is played before the transfer happens.

- **Connect Timeout** - Enter a positive integer.

  This number is the number of seconds that the system waits for a response from the dialed party. If the system receives no response within this time, control of the call returns to the application. In this case, the system also returns a value of **noanswer** to the **value** field of the Bridged Transfer variable.

- **Maximum Time** - Enter a positive integer.

  This number is the maximum number of seconds that a transferred call can last. When a transferred call reaches this number of seconds, the system ends the transfer and returns control of the call to the original application. In this case, the system also returns a value of **maxtime_disconnect** to the **value** field of the Bridged Transfer variable.

  A value of zero (**0**), which is the default, means the system imposes no time limit on call transfers.

- **Destination Number** - Enter the telephone number to which the system will transfer callers.

  Use this when you want to “hard code” the number. For example, if you wanted to transfer the caller at this point to an operator, you might enter 00, the number to dial for the system operator.

  If you use this field, you cannot use the **Destination Variable** field.

- **Destination Variable** - Select the variable that contains the number to which the system will transfer callers. Use this when you want to allow for dynamic transfers. In dynamic transfers, for example, callers can specify the telephone number to call, and that number is stored in a variable. In other dynamic transfers, the system obtains the destination telephone number from some other source and stores it in a variable.

  If the variable you select here is a complex variable, you must also select a **Destination Variable Field**.

  If you use this field, you cannot use the **Destination Number** field.

- **Destination Variable Field** - If the selected destination variable (see previous item) is a complex variable, you must select a field from the drop-down list. If the destination variable is a simple variable, this option is inactive.

- **AAI Data** - Enter application-to-application literal data that you want to pass from this application to the receiving end of the call transfer. This option passes the raw data just as you enter it in this field.

- **AAI Variable** - Select the variable that contains the application-to-application data you want to pass to the receiving end.

  If the variable you select here is a complex variable, you must also select an **AAI Variable Field**.

- **AAI Variable Field** - If the selected **AAI Variable** is a complex variable, select the field that contains the application-to-application data you want to pass to the receiving end.

  If the **AAI Variable** is a simple variable, this drop-down list is not populated.

- **Bind Name to Node** – (true/false). When true, the name of the variable item is bound to the name of the node. When the parent node is renamed, the name of the variable item is renamed as well; keeping the name of the variable in sync with the node name.

Capture Expression 

Type

Call Flow item

Available from

- Announce Node
- Prompt and Collect Node
- Form Node
- Menu Node
- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)

Purpose

The purpose of the Capture Expression item is to capture the value of ECMA script objects from the Voice XML page and store them in a Dialog Designer session variable.

Behavior

The Capture Expression item is used in a few different ways:

- **App Root** - On the AppRoot, Capture Expression is used to capture ECMA script values as well as supplemental session variables when the voice dialog starts up. It can be used for capturing the value of SIP headers, for example, or other VoiceXML platform variables that are not automatically captured by Dialog Designer when a dialog starts.

  Refer to the documentation for your voice portal platform for further details on headers and other variables that are exposed.

- **Event handlers** - Capture Expression can be added to goto items used in Event handlers for capturing the event type and event message. The event type/message could then be used in another section of the callflow for handling errors differently.
- **Forms/Menus** - Can be added to goto items on links for capturing values from the **application.lastresult$** variable such as capturing the **utterance** that activated the link, or the ASR **confidence**, etc.

Note:
Dialog Designer does not validate the ECMA script expression that is entered. It is expected that the designer has consulted the VXML specification and understand which ECMA script objects are valid at different places of the generated VXML page.

Properties

- **Variable** - The variable that will store the captured data. Select the variable you want to use. If the variable you select is a complex variable, you must also select a **Variable Field**.
- **Variable Field** - Select the variable field you want to use. The contents of this list depend on which variable you selected in the **Variable** property:
- **Expression** - The ECMA script expression that will be evaluated by the VoiceXML browser. The result is stored in the variable. The ECMA script object must be simple (for example, strings or numbers).

Catch (Exception) 

Type

Data Node exception handler

Available from

Data Node

Purpose

To catch Java exceptions that are thrown at runtime.

Behavior

Try/Catch items in the Data Node will *try* to execute items under the Try item. To handle exceptions, add Catch (Exception) items (under the parent Try). For every Try item, you need at least one Catch.

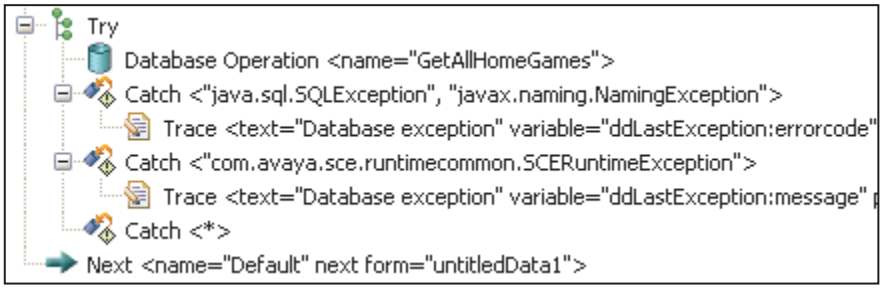
By default, Catch (Exception) will catch *all* exceptions. Catch (Exception) can also be used to catch specific exception types for handling specific errors differently. Comma separated exception types can also be specified for catching different exception types, but handling them in the same way.

List of common exceptions are provided in the Properties view. Currently this list contains:

- SQLException
- IOException
- SCERuntimeException

A new variable called **ddLastException** automatically captures the exception caught by a Catch item (includes the following data members: errorcode, message, object, stacktrace, and type).

Following is an example of the Try/Catch mechanism.



Catch (VXML Events)

Type

Event handler item

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Menu Node
- Form Node

Purpose

The **Catch** event handler is a generic handler for events you create. Catch (VXML Events) catches VXML events that are thrown in the application or by the VXML platform.

Note:
Dialog Designer has built-in event handlers for most of the default VoiceXML events. The **Catch** event handler is designed to handle all other, custom, events. For more information about the built-in event handlers, see:

- No Input
- No Match
- On Disconnect

Behavior

Before you can use the **Catch** event handler, you must create the event for it to catch. You must also, somewhere in your application, within the scope of this event handler, use a **Throw** item.

The **Catch** event handler responds to a specified event anywhere within the scope of the handler. For example, if you place the **Catch** event handler in the **AppRoot** node, it acts as a global event handler for the specified events. It can catch these specified events no matter where in the application the events are thrown. If you place the **Catch** event handler in another node, however, it can catch the specified event only if it is thrown while in that node. If you place the **Catch** event handler underneath a node item, it can catch the specified event only if it is thrown within that node item.

A **Catch** event handler at the node item level overrides a **Catch** event handler at the node level. Likewise, a **Catch** event handler at the node level overrides a **Catch** event handler at the **AppRoot**, or global, level.

For more information about creating events, throwing events, and catching events, see Working with Event Types.

Examples for handling events:

1. Catch (event=”myEvent”), Return Event

   In this scenario, when the “myEvent” event is caught, the module will exit, returning control back to the calling application, and it will return the “myEvent” that was caught.

   In other words, it propagates or re-throws the event to let the calling application handle the event.

2. Catch (event=”error.runtime”), Return Event, Throw (event=”myReallyBadEvent”, message=”Something happened...”)

   In this scenario, the application catches an “error.runtime” event and the module will exit returning control back to the calling application.

   Instead of propagating the “error.runtime” event, however, it returns “myReallyBadEvent” instead, with the message “Something happened...”. The calling application can catch the “myReallyBadEvent” event, but does not know that the cause of the event was an “error.runtime” event.

3. Catch (“error.badfetch”), Exit

   In this case, the application catches an “error.badfetch” event and then exits the application. Exiting the application will essentially disconnect the caller and terminate the session.

   It does not matter if it is a module or a stand-alone application, the application will end. This is typically used when there is some fatal error with the application or system.

Properties

- **Event** - Select the event that you want the **Catch** event handler to handle.

  For the **Catch** event handler to work, you must specify the event for it to catch when the event is thrown. You must also use one or more of the following options with it:

- **Prompt item** - This item plays the designated prompt to the caller before the application continues.
- **Goto item** - This item directs the application to the designated node.
- **Throw item** - Throws an event that you choose. Select a user-defined event that you have created.

  By throwing an event within an event handler, you can transfer control to a different part of the application, for example, to the **AppRoot** node. For more information about throwing events and handling events, see Throw or Working with Event Types.

Choice 

Type

Menu item

Available from

Menu Node

Purpose

When you use a Menu node in your call flow, you must offer callers at least one choice, or it does not make sense to use a menu. **Choice** items exist for the purpose of providing callers with options on a menu, from which they can choose.

Behavior

The **Choice** item uses either a spoken reply or a DTMF response to direct the call flow based on the response of a caller. Each **Choice** item in a Menu node represents a single option available to the caller. So, if you want to offer callers a menu with four options, you must use four **Choice** items, one for each option.

If you want to use ASR, you must assign a grammar to each **Choice** item.

Note:
When you assign grammars to choices, keep in mind that any recognized response in a grammar will activate the choice to which it is assigned. For this reason, Avaya strongly recommends that you use a separate grammar for each choice in the menu. Each of these grammars can contain only one or two words for recognition. For more information about creating and using grammars, see Working with Grammars.

Properties

Enter values for the following properties:

- **Name** - Enter a descriptive name to indicate what this choice represents.
- **Accept Choice** - Select whether you want the choice recognition to be:
  - **Exact** - (the default) In this case, the system recognizes the response of the caller successfully only if the response is exactly what the system expects. For example, if the system is set to respond to “Weekly News in Review” exactly, the system recognizes only that exact phrase with all words in the proper order.
  - **Approximate** - In this case, the system can recognize a range of responses, usually a subset of the expected response. For example, if the system is set to respond to “Weekly News in Review” approximately, the system can recognize “Weekly News,” “News Review,” and “Weekly Review” all as valid responses.

  Note:
  This option does not work well with the Microsoft Speech SDK, which Dialog Designer uses to test ASR in applications. For this reason, you might not be able to test this option until you deploy the application to a live system.
- **Next Form** - Indicates what node this choice goes to when selected. You can either:
  - Select the node you want this choice to go to.
  - Use the graphic Connector tool in the main.flow window to connect this choice to the desired node.

- **DTMF** - (optional) Enter in this field the DTMF key press or key press combination that the caller can use to select this choice.

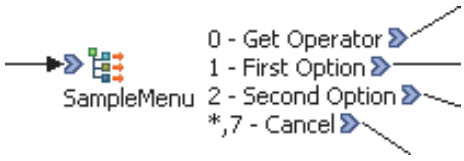
Note:

If you designate a key press or series of key presses, Dialog Designer displays these key presses in the Call Flow Editor main workspace as part of the branch arrows that lead away from the node. If multiple key presses are indicated for a menu option, Dialog Designer lists the key presses separated by commas. For an example, see Figure 1.

Tip:

It is always a good idea to give callers the option to use either a spoken reply or a DTMF key press to select a choice. For example, you can create a prompt for a choice that says, “To hear about your savings options, press one or say "savings." Then, when setting up this Choice item, you both use an ASR grammar for “savings” and enter **1** in the DTMF field.

Figure 1: Sample Menu node, showing DTMF menu options



Column Operand

Type

Database Operation Condition Operand

Available From

Database Operation Editor Predicate Tab

Purpose

For specifying the table name and the column name on the left side or right side of a condition. It applies to both Simple and Join conditions.

Behavior

When adding a Simple condition item, both the Column item and Comparison item are automatically added inside the Simple condition item. Similarly, adding a Join condition would automatically create both Column items and the Comparison item inside it.

Properties

- Table Name: choose the table that has been added to the database operation.
- Column Name: choose the column from the list for the table selected.

Comparison Operator 

Type

Database Operation Condition Operator

Available From

Data Node

Database Operation Editor Predicate Tab

Purpose

For setting up the comparison between the left side and the right side of a condition. It is needed when using both simple and join conditions

Behavior

When adding a Simple condition item, both the Column item and Comparison item are automatically added inside the Simple condition item. Similarly, adding a Join Condition would automatically create both Column items and the Comparison item inside it.

Properties

- **Operator** – Choose from (=, <, <=, >, >=, <>, LIKE, NOT LIKE)

Complex Variable 

Type

Variable

Available from

Variables Editor

Purpose

A complex variable is a variable that has several attributes, called *fields*. Complex variables make it possible to use a single variable to pass along multiple values. To be valid, a complex variable must have at least one Field assigned to it.

Behavior

A complex variable must have at least one variable field to be valid. For more information about complex variable fields, see Field.

When you create a complex variable, Dialog Designer automatically assigns a temporary name to it and adds one **Field** item. You can rename both the variable and the field. You can also add as many additional **Field** items as you want.

In some cases, Dialog Designer creates complex variables automatically when nodes or items are used in the call flow. For example, when you place a Bridged Transfer item in the call flow, Dialog Designer automatically creates a complex variable with the same name as the item.

For more information about the use of variables in Dialog Designer, see Working with Variables.

Properties

- **Name** - Enter in this field the name to identify the variable.

Note:
This name must follow conventions for naming Java components as described in Naming Java Components.

Condition

This item has been deprecated.

- In the Data Node, use right-click to replace with If+Next.
- In the Prompt Editor, use right-click to remove the condition parent item. The If and Else child items will be preserved.

Consultation Transfer

Type

Form item

Available From

- Announce Node
- Blind Transfer Node, pre-populated by default
- Form Node

Note:
Although the **Consultation Transfer** item appears on palettes for other nodes, such as the Prompt and Collect node, the Bridged Transfer node, and the Record node, it is not actually available for use in those nodes without destroying their intended functionality. It is intended primarily for use only in the Consultation Transfer Node node.

Purpose

Consultation Transfer is used for transferring a call to another destination without releasing the call from the voice platform until after the call is successfully transferred. If the connection cannot be established to the callee, then the caller is returned back to the speech application which can attempt to handle the call in other ways.

Behavior

A Consultation Transfer is similar to a Blind Transfer except that the outcome of the transfer call setup is known and the caller is not dropped as a result of an unsuccessful transfer attempt. When performing a consultation transfer, the platform monitors the progress of the transfer until the connection is established between caller and callee.

If the connection cannot be established (e.g., no answer, line busy, etc.), the session remains active and returns control to the application. As in the case of a blind transfer, if the connection is established, the interpreter disconnects from the session, connection.disconnect.transfer is thrown, and document interpretation continues normally.

If the connection to the callee is successfully established, then the call is transferred and the application platform releases the call and the speech application ends (the application may execute further to perform logging, resource cleanup, etc., but the caller is no longer connected to the application).

If successful, the application will get a **connection.disconnect.transfer** event and the VXML session ends; or it continues with session termination operations. If it fails, then the variable item will store the error code and the application can then evaluate the error and perform the appropriate action. See the VXML specification for return codes. Define event types using the Event Type Editor, and use the VXML event handlers to Catch (VXML Events) the event.

Properties

- **Transfer Audio** – - Select the prompt to use for the call transfer.

  The **Transfer Audio** prompt must be a transitional audio prompt. This transitional audio prompt is the audio (*.wav) file that is played to callers while the call is being transferred. For more information about transitional audio prompts, see About Transitional Audio Prompts.

- **Connect Timeout** – Maximum number of seconds to allow before the terminal connection is established.

- **Name** - Enter the name you want to assign to the item and the corresponding variable. If you change the name of this item in the **Properties** view, Dialog Designer automatically changes the name of the variable as well.

  Note:
  This name must follow conventions for naming Java components as described in Naming Java Components.

- **Destination Number** - Enter the telephone number to which the system will transfer callers.

  Use this when you want to “hard code” the number. For example, if you wanted to transfer the caller at this point to an operator, you might enter 00, the number to dial for the system operator.

  If you use this field, you cannot use the **Destination Variable** field.

- **Destination Variable** - Select the variable that contains the number to which the system will transfer callers.

  Use this when you want to allow for dynamic transfers. In dynamic transfers, for example, callers can specify the telephone number to call, and that number is stored in a variable. In other dynamic transfers, the system obtains the destination telephone number from some other source and stores it in a variable.

  If the variable you select here is a complex variable, you must also select a **Destination Variable Field**.

  If you use this field, you cannot use the **Destination Number** field.

- **Destination Variable Field** - If the selected destination variable (see previous item) is a complex variable, you must select a field from the drop-down list. If the destination variable is a simple variable, this option is inactive.

- **AAI Data** - Enter application-to-application literal data that you want to pass from this application to the receiving end of the call transfer. This option passes the raw data just as you enter it in this field.

- **AAI Variable** - Select the variable that contains the application-to-application data you want to pass to the receiving end.

  If the variable you select here is a complex variable, you must also select an **AAI Variable Field**.

- **AAI Variable Field** - If the selected **AAI Variable** is a complex variable, select the field that contains the application-to-application data you want to pass to the receiving end.

  If the **AAI Variable** is a simple variable, this drop-down list is not populated.

- **Bind Name to Node** – (true/false). When true, the name of the variable item is bound to the name of the node. When the parent node is renamed, the name of the variable item is renamed as well; keeping the name of the variable in sync with the node name.

Blind Call (CTI)

Type

CTI connector item

Available From

Data Node (CTI section)

Purpose

Used to transfer the caller to a destination number without knowing the call results. If the transfer fails, the call is lost and the application cannot retrieving it.

Behavior

When this node is invoked, the caller is put on hold, then the number to dial out is attempted. The call will be transferred to the destination automatically in all cases except call failure where the switch will not allow the transfer. In the case of failure, the call is dropped. If successful and UUI data has been indicated, it will be sent to the destination.

Note:
It is recommended to use a **Try/Catch** around the operation to capture catastrophic errors. In all cases, the call is terminated if it fails because it is a blind call.

Properties

- **Name** - Unique name of the node. Will be used to create a call info variable that holds call information about the newly dialed call.
- **Caller Call Info Variable** - Call info variable that represents the current caller. Typically, this is the cticallinfo variable.
- **Dial Number Variable/Field** - Variable and field used to hold the number to dial.
- **UUI Variable/Field** - Variable and field used to hold the UUI information that is passed in with the new call. UUI is user to user information and allows you to pass up to 96 bytes of character data.

Call Info (CTI)

Type

CTI connector item

Available from

Data Node only

Purpose

In speech applications that use computer telephony integration (CTI) to extend call control capabilities, the **Call Info** item collects a variety of data about the call and assigns this data to variable fields. You can then use these variable values to define the way the call is handled in the CTI environment.

Behavior

Before you can use this or any other CTI connector item, you must enable CTI connectors for Dialog Designer. For more information about the ability of Dialog Designer to interact with CTI, and to enable CTI in your Dialog Designer applications, see About CTI Connectors.

When you place the **Call Info** item in a Data node, Dialog Designer automatically creates a complex variable with the same name as the item, and populates it with the following fields:

- **ani** - Contains the telephone number the caller is calling from, as provided by the Automatic Number Identification (ANI) service.
- **callid** - Contains the call ID number for the call, as assigned by the telephony server.
- **dnis** - Contains the telephone number the caller dialed, as provided by the Dialed Number Identification Service (DNIS).
- **stationextension** - Contains the number of the port on the switch that the call came into.
- **ucid** - Contains the universal call ID number.

  The universal call ID number is an identification number that the network assigns to the call when it enters the network. The system uses this number to track the call no matter what other locations or sites in the network the call might be routed to.
- **uui** - Contains any user-to-user information (UUI) that might exist with the call.

  UUI is text-based data that accompanies the call. For example, during the course of a call, you might have collected information from the caller such as an account number, customer preferences, customer identification data, and so forth. You can use CTI transactions to pass this data to a call center agent, where this data can be used to populate a data window on the agent's computer screen.

Note:
Dialog Designer does not display the **Call Info** variable and its fields until you save the application. After you save the application project, the variable and its fields are available from the Variable Editor. See Working with Variables.

cticallinfo is populated before your first node runs. This palette item is used mostly for submodules. You have to pass in a call ID to obtain the **Call Info** data. When you run a submodule, the **cticallinfo** variable is not populated, however, so **cticallinfo** is necessary there.

Properties

- **Call ID Variable** - Select the variable to use. Avaya recommends that you use the **cticallinfo.callid** field that the system creates when you enable CTI. Typically this is passed to a module as an input value.
- **Name** - Enter in this field the name you want to assign to the **Call Info** item and its associated variable.

  Dialog Designer assigns a default name automatically when you place the **Call Info** item in the node, and you can leave it as is or rename it.

Note:
This name must follow conventions for naming Java components as described in Naming Java Components.

Conference (CTI)

Type

CTI connector item

Available from

Data Node

Purpose

In speech applications that use computer telephony integration (CTI) to extend call control capabilities, the **Conference** item makes it possible to take two telephone calls on the same port and merge them into a single conference call.

This additional caller is added to the conference with announcement (announced) to the existing callers. Sometimes, this type of call operation is also called *three-step conference*.

Behavior

Before you can use this or any other CTI connector item, you must enable CTI connectors for Dialog Designer. For more information about the ability of Dialog Designer to interact with CTI, and to enable CTI in your Dialog Designer applications, see About CTI Connectors.

Before you can conference a call, you must place the incoming call on hold. For information about placing calls on hold, see Hold (CTI). You must then use a Dial (CTI) item to dial the number to conference the incoming call with. The call you are trying to conference with, then, is the active call. After the system successfully establishes a connection with the number dialed, then it merges the call on hold with the new connection.

To extend the conference beyond three callers and up to a maximum of six callers, you can add callers using either additional Conference (CTI) items or Conference Party (CTI) items. Beyond six callers, the voice portal/IR port drops out and no additional callers are allowed to be added to the conference.

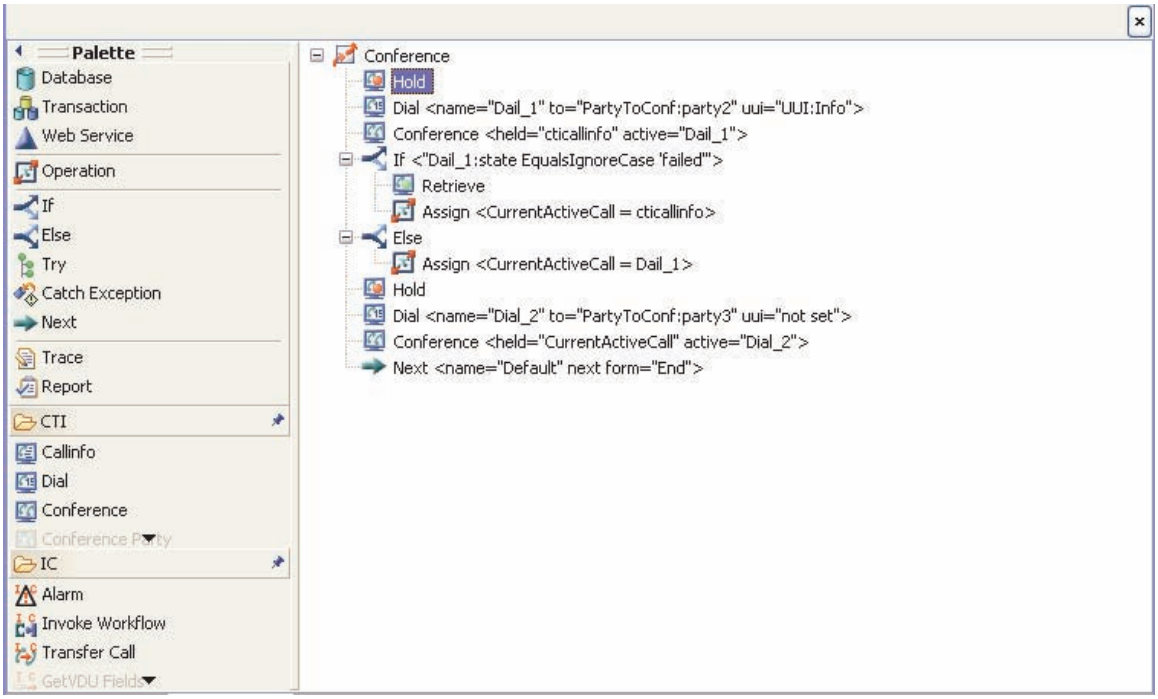
The active **cticallinfo** variable is the one that is used for the last Dial (CTI). The active **CallInfo** variable must be used when adding another caller to the conference using Conference (CTI).

Return status' are found in the state field of the active **cticallinfo** variable. There are two possible states: conferenced or fail.

See About CTI Connectors.

Following is an example showing how to conference two parties. CurrentActiveCall is a variable that is defined with the same structure as the **cticallinfo** variable to track the current active call.

CTI Conference Example



Properties

- **Held Call ID Variable** - Select the variable that contains the name of the **CallInfo** variable associated with the call that is on hold. You must also select the **callid** field for the **Held Call ID Variable Field**.
- **Held Call ID Variable Field** - Select the **callid** field item.
- **Active Call ID Variable** - Select the variable that contains the name of the **Dial** variable associated with the call that is being dialed. You must also select the **callid** field for the **Active Number Variable Field**.

- **Active Call ID Variable Field** - Select the **callid** field item.

Conference Party (CTI) 

Type

CTI connector item

Available From

Data Node

Purpose

In speech applications that use computer telephony integration (CTI) to extend call control capabilities, the **Conference Party** item makes it possible to add an additional caller to an existing conference.

This additional caller is added to the conference without announcement (unannounced) to the existing callers. Sometimes, this type of call operation is also called *single-step conference*.

Behavior

Assuming that you already have an existing conference call, Conference Party (CTI) is used to add an additional party to the call. This is sometimes called a “single-step conference” because a party is conferenced into a Call directly. The telephone address string provided as the argument must be complete and valid.

Properties

- **Party Number Variable** – Holds the predefined variable (phone number) of the conference that the additional party is trying to conference into.
- **Party Number Variable Field** – Select the **Party Number Variable** from the predefined list of fields.

Consultation Call (CTI) 

Type

CTI connector item

Available From

Data Node (CTI section)

Purpose

Used to transfer the caller to a destination number while maintaining control of the call. If the transfer is not successful, the caller will be taken off from hold and the call results can be analyzed for further action.

Behavior

When this node is invoked, the caller is put on hold, then the number to dial out is attempted with a newly dialed call. If the call is successfully transferred, the caller will be transferred as well as any UUI data that has been indicated. If the call fails, the caller is taken off from hold and is able to interact with the application again. If a call enters a queue, it will be automatically transferred when the queued event occurs.

When a consultation call item is used, a call info variable with the name of that consultation call item will be created. The status of the call can be determined from the <Name>.state field where Name is the unique name property given to the consultation call item.

The status of the call can be determined from the **<Name>.state** field where Name is the unique name property given to this node and created as a call info variable.

Note:
It is recommended to use a **Try/Catch** around the operation to capture catastrophic errors.

In terms of call states, for the caller on hold, the states include: **transferred**, **disconnected**, or **established**. The **established** state means the caller was not able to transfer and has been taken off hold for an alternative operation. If the “Transfer on Ring” property is set, the noanswer state is not encountered because the operation will not check to determine if the callee has picked up.

Properties

- **Name** - Unique name of the node. Will be used to create a call info variable that holds call information about the newly dialed call and to allow analysis of call.
- **Caller Call Info Variable** - Call info variable that represents the current caller. Typically, this is the cticallinfo variable.
- **Dial Number Variable/Field** - Variable and field used to hold the number to dial.
- **UUI Variable/Field** - Variable and field used to hold the UUI information that is passed in with the new call. UUI is user to user information and allows you to pass up to 96 bytes of character data.
- **Ring Max** - Maximum number of rings before determining the call is ring no answer. Each ring will take approximately 6 seconds.
- **Transfer on Ring** - When this property is set to true, the consultation call will transfer the call when it detects a ring event. All errors that occur before the ring will be handled properly. The ring timer does not have any affect when this flag is set since the call is transferred on first ring.

Dial (CTI) 

Type

CTI Item

Available from

Data Node only

Purpose

In speech applications that use computer telephony integration (CTI) to extend call control capabilities, the **Dial** item requests that a call be placed from the IVR system to a destination telephone number.

Behavior

Before you can use this or any other CTI connector item, you must enable CTI connectors for Dialog Designer. For more information about the ability of Dialog Designer to interact with CTI, and to enable CTI in your Dialog Designer applications, see About CTI Connectors.

Note:
During a CTI dial operation, when a queue is hit, the CTI connector will stop the ring timer and return a “queued” state back to the application. The application should check for that state and transfer the call. In the case of a CTI Consultation Call (CTI), the call will automatically be transferred when it hits a queue.

Note:
It is recommended to use a **Try/Catch** around the operation to capture catastrophic errors.

When you place the **Dial** item in a Data node, Dialog Designer automatically creates a complex variable with the same name as the item, and populates it with the following fields:

- **ani** - Contains the telephone number the original caller is calling from, as provided by the Automatic Number Identification (ANI) service.
- **callid** - Contains the call ID number for the original call, as assigned by the telephony server.
- **dnis** - Contains the telephone number the original caller dialed, as provided by the Dialed Number Identification Service (DNIS).
- **state** - Returns the current state of the call. Possible values include the following values: **established**, **disconnected**, **failed**, **queued**, **unreachable**, **noanswer**, and **busy**.
- **stationextension** - Contains the number of the original port on the switch that the call came into.

- **ucid** - Contains the universal call ID (UCID) number. This field is only valid if it is enabled on the PBX/switch.

  The universal call ID number is an identification number that the network assigns to the call when it enters the network. The system uses this number to track the call no matter what other locations or sites in the network the call might be routed to.

- **uui** - Contains any user-to-user information (UUI) that might exist with the call.

  UUI is text-based data that accompanies the call. For example, during the course of a call, you might have collected information from the caller such as an account number, customer preferences, customer identification data, and so forth. You can use CTI transactions to pass this data to a call center agent, where this data can be used to populate a data window on the agent's computer screen.

Note:
Dialog Designer does not display the **Dial** variable and its fields until you save the application. After you save the application project, the variable and its fields are available from the Variable Editor. For more information about variables, see Working with Variables.

This item is used in conjunction with a Transfer (CTI) or Conference (CTI) operation. See Using CTI Connectors in Applications.

Properties

- **Name** - Enter in this field the name you want to assign to the **Dial** item and its associated variable.

  Dialog Designer assigns a default name automatically when you place the **Dial** item in the node, and you can leave it as is or rename it.

Note:
This name must follow conventions for naming Java components as described in Naming Java Components.

- **Dial Number Variable** - Select the variable that contains the telephone number for the system to dial.

  If this variable is a complex variable, you must also select a **Dial Number Variable Field**.

- **Dial Number Variable Field** - If the **Dial Number Variable** is a complex variable, select the variable field that contains the telephone number for the system to dial.

- **UUI Variable** - Select the variable that contains the user-to-user information (UUI) that you want to pass to the destination telephone number.

  If this variable is a complex variable, you must also select a **UUI Variable Field**.

  This variable can contain up to 96 bytes of data.

- **UUI Variable Field** - If the UUI Variable is a complex variable, select the UUI variable field to which you want to pass to the destination telephone number.

- **Ring max** - Enter in this field the number of rings the system should wait for the destination telephone number to respond before the system terminates the attempt. Whenever the destination number exceeds this number of rings, then the system returns a state of **noanswer** and moves to the next item in the subflow.

Disconnect (CTI)

Type

CTI connector item

Available from

Data Node only

Purpose

In speech applications that use computer telephony integration (CTI) to extend call control capabilities, the **Disconnect** item terminates a call that has been transferred or conferenced using CTI.

Behavior

Before you can use this or any other CTI connector item, you must enable CTI connectors for Dialog Designer. For more information about the ability of Dialog Designer to interact with CTI, and to enable CTI in your Dialog Designer applications, see About CTI Connectors.

When you place the **Disconnect** item in a Data node, Dialog Designer uses it to terminate a CTI connection that has been established either in this node or another.

Properties

- **Call ID Variable** - Select the variable to use. Avaya recommends that you use the **cticallinfo** variable that the system creates when you enable CTI.

Hold (CTI)

Type

CTI connector item

Available from

- Data Node

Purpose

In speech applications that use computer telephony integration (CTI) to extend call control capabilities, the **Hold** item puts a caller on hold while the call is transferred or conferenced.

Behavior

Before you can use this or any other CTI connector item, you must enable CTI connectors for Dialog Designer. For more information about the ability of Dialog Designer to interact with CTI, and to enable CTI in your Dialog Designer applications, see About CTI Connectors.

When you place the **Hold** item in a Data node, the Dialog Designer application places the original call in a hold state until one of the following is true:

- The call transfer or conference has been established.
- The system determines that the transfer or conference cannot be established and a Retrieve (CTI) item takes the call out of the hold state.

Properties

- **Call ID Variable** - Select the variable to use. Avaya recommends that you use the **cticallinfo** variable that the system creates when you enable CTI.

Retrieve (CTI) 

Type

CTI connector item

Available from

- Data Node

Purpose

In speech applications that use computer telephony integration (CTI) to extend call control capabilities, the **Retrieve** item places a caller that has been on hold back in an active state.

Behavior

Before you can use this or any other CTI connector item, you must enable CTI connectors for Dialog Designer. For more information about the ability of Dialog Designer to interact with CTI, and to enable CTI in your Dialog Designer applications, see About CTI Connectors.

When you place the **Retrieve** item in a Data node, the Dialog Designer application takes a CTI call that is in a Hold (CTI) state and places it back in an active state within the application. This item is usually used when a CTI call transfer or conference has been attempted but not successfully established.

Properties

The **Retrieve** item has no properties that can be modified.

Transfer (CTI) 

Type

CTI connector item

Available from

Data Node only

Purpose

In speech applications that use computer telephony integration (CTI) to extend call control capabilities, the **Transfer** item makes it possible to transfer not only the call itself, but also associated data.

Note:
This type of transfer is different from the standard call transfer capabilities of Dialog Designer. The two standard call transfer types, Blind Transfer and Bridged Transfer, both transfer only the call. They do not provide the capability to transfer data with the call.

Behavior

Before you can use this or any other CTI connector item, you must enable CTI connectors for Dialog Designer. For more information about the ability of Dialog Designer to interact with CTI, and to enable CTI in your Dialog Designer applications, see About CTI Connectors.

Note:
Support for call transfer functions can vary from one IVR system to another. For more information and details about support for transfer functions on your system, see the documentation for your IVR system.

Before you can transfer a call and its associated data to another number, you must place the incoming call on hold. For information about placing calls on hold, see Hold (CTI). You must then use a Dial (CTI) item to dial the number to transfer to. The number you are trying to transfer to, then, is the active call. After the system successfully establishes a connection with the number dialed, then it connects the call on hold with the new connection.

For more information, including a suggested procedure to create a conference call using CTI connectors in Dialog Designer, see Constructing a Call and Data Transfer Using CTI Connectors.

Properties

- **Call ID Variable** - Select the variable to use. Avaya recommends that you use the **cticallinfo** variable that the system creates when you enable CTI.

Database Operation 

Type

Node item

Available from

Data Node

Purpose

The **Database** item makes it possible to use a predefined database operation. The **Database** item in itself only invokes the database operation and tells the call flow where to use it.

Behavior

Before you can use the **Database** item in the Data node, you must have a data source defined for the application. For more information about defining and using data sources, see Configuring Data Sources.

You must also have at least one database operation created for the application. For information about setting up and using database operations, see Using the Database Operation Wizard to Create a Database Operation File.

The way the database operation actually behaves in the Data node depends on how the database operation is defined. When you place the **Database** item in the Data node flow, all Dialog Designer does is invoke the selected database operation.

Properties

- **Name** - Select the database operation you want to invoke.

Database Transaction

Type

Node item

Available from

Data Node

Purpose

Used for grouping multiple database operations in a single transaction. If one step fails in a transaction, all the changes are rolled back. Changes are only committed when all operations within the transaction succeed.

Behavior

By combining a Database Operation item and Operation item into a Data Node, the items can be grouped within the Database Transaction node to make a single transaction.

Properties

None.

Else 

Type

Condition item

Available from

Any level tab in the Prompt File Editor

Data Node in the Call Flow Editor

Purpose

The Condition item makes it possible to select and play different prompt or data segments based on the condition being tested. It works in conjunction with If and **Else** items to test conditions. The **Else** item defines a default for cases where no conditions of an **If** item are met.

In the Call Flow Editor, the Else item is used to conditionally execute different operations based on the evaluation of logical statements. It works in conjunction with an If item to execute operations when the “If” condition evaluates to a “false” value.

Behavior

The **Else** item cannot be used independently but must be used in conjunction with the If item.

If there are multiple **If** items, they are all tested for first. If none of the conditions being tested for are met in any of the **If** items, the system returns the result assigned to the **Else** item.

Properties

The **Else** item has no inherent properties. It can only have one or more prompt segments assigned to it.

Emphasis 

Type

SSML item

Available from

Any level tab in the Prompt File Editor. For more information about the Prompt File Editor, see Using the Prompt File Editor.

Purpose

In natural speech, some words are emphasized more than others. That is, we place a greater stress on some words than others when we talk. The **Emphasis** item causes the speech synthesis engine to place emphasis on selected text to be rendered.

Note:
The **Emphasis** item and its properties function correctly only with SSML-compliant speech synthesis engines. The Microsoft Speech SDK, which is used by Dialog Designer during application simulation, is *not* an SSML-compliant speech synthesis engine, so any settings you make with this item are ignored. For more information about the SSML standard, see the Speech Synthesis Markup Language (SSML) Version 1.0 W3C Recommendation.

Behavior

When you place an **Emphasis** item in a prompt, you can choose to either:

- Place it at the root level of the prompt.
- Attach it to a specific TTS or text variable segment.

If you place the **Emphasis** item at the root level of the prompt, the setting of the **Emphasis** item affects all the TTS and text variables segments from that point forward. It remains in effect for the rest of the prompt or up to the point where you place another **Emphasis** item at the root level.

If you attach the **Emphasis** item to a particular TTS or text variable segment, the setting of the **Emphasis** item affects only that TTS or text variable segment.

So, for example, to create a TTS sentence that says, “That car is a *beautiful* one,” with a strong emphasis on the word “beautiful”:

1. Create a prompt with three TTS prompt segments:
   - “That car is a”
   - “beautiful”
   - “one”
2. Attach an **Emphasis** item to the second segment with a **Level** setting of **strong**.

Properties

- **Level** - Select one of the following settings:

Emphasis Setting	Description
strong	Applies a great amount of emphasis to the selected text.
medium	(default) Applies a moderate amount of emphasis to the selected text.
none	Applies no emphasis to the selected text. You can use this setting, in effect, to cancel a prior **Emphasis** item setting.
reduced	In effect, acts as the opposite of emphasizing a word. This setting actually reduces the normal amount of emphasis on the selected text. For example, “going to” with a setting of **reduced** might be rendered as “gonna.”

Note:
The actual amount of emphasis applied to the text depends on the speech synthesis engine. The amount of emphasis also depends on the language being spoken. This is because each language indicates emphasis differently, based on a combination of pitch changes, timing changes, loudness, and other acoustic differences.

Exit 

Type

Event handler

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Form Node

Purpose

When events are caught (at the **Field** level, **Form** level or **AppRoot**), there are various ways to handle the event (such as re-playing prompts, going to different forms, etc.). In the case where there are more serious errors, consider using the **Exit** event handler to exit the application.

Behavior

The exit item is used to disconnect the caller and terminate the session. The exit item is intended to be used when “fatal errors” occur and there is no recovery.

A typical use of the Exit item is as a child of Catch (“error.badfetch”). Usually, when the application encounters these “error.badfetch” errors, it means that the application server is unavailable or the caller’s session on the application server is no longer valid.

In this scenario, it is not possible to fetch other VXML pages to handle the event because this usually results in additional “error.badfetch” events. To prevent endless bad fetch loops, using Exit will allow the VXML platform to release the call.

See Catch (VXML Events) for other ways to handle VXML events.

Properties

- **Threshold** - A variable that must have a positive integer value.

Note:
This field is available only when the **Exit** item is placed in an event handler or a node.

This property is used to disable the **Exit** item until this value is met. For example, if the **Exit** property is set to 3, the system ignores the Throw item the first two times the specified event is encountered. Only the third time that the event is encountered does the system actually throw the event.

Expression

Used to embed an ECMA script expression within a prompt. This can be used for playing event details or other ECMA script values.

For example, when a prompt is played in an Event handler, the prompt may use the Expression element to play “_message” (the event message).

Using Expression elements in a prompt requires knowledge of the valid ECMA script objects in a VXML document. The value of the Expression property is literally embedded in the generated VXML, so if the Expression is invalid, there may be VXML parse or semantic errors.

External Property

Type

Variable item

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Form Node

Purpose

The W3C VoiceXML 2.0 Recommendation uses the <property> element to set values that affect system behavior. These values can include properties like timeouts, recognition processes, and caching policies. Dialog Designer uses the **Property** item to set many of these common properties, but some IVR systems make it possible to set other properties that are not part of the standard Dialog Designer set.

In addition to the standard properties that Dialog Designer supports, some IVR systems have custom properties available. Use the **External Property** item to set those properties on the IVR system.

Note:
For more information about the standard properties supported in Dialog Designer, see Property.

Behavior

The **External Property** item takes a named property value and passes it to the IVR system. The format, content, and resultant behavior of the value depend on what the IVR system expects and can process.

An example of using an **External Property** item can be described in the usage of the VoiceXML property **fetchtimeout** as an External Property item. In this case, the voice browser (AAS) waits a given amount of time (that is, the fetch timeout time) to get a page response back from the application. This timeout is configurable on the platform and applies to all fetches done for all applications. Some applications may exceed this timeout during long operations. Such operations could be database access, a CTI dial operation that has a long ring timeout, Web service calls, etc.

Instead of changing the platform timeout, set the **fetchtimeout** property in a node or in the AppRoot (for an application-wide change) and change the timeout to a more appropriate value. For example, set Name to fetchtimeout and value to 30 seconds (15 seconds more than the default timeout).

Note:
A very long **fetchtimeout** can have a negative effect on the efficiency of the browser in cases where the application has a catastrophic error and the browser is waiting for a page response or exceeding the timeout.

Properties

- **Name** - Enter the name to identify the property. This name is sent, along with the **Value**, to the IVR system on which the property is to be set.
- **Value** - Enter the exact value to be sent to the IVR system. This must be in the form and format that IVR system expects and can process.
- **Variable** - Select the variable you want to use. If the variable you select is a complex variable, you must also select a **Variable Field**.

- **Variable Field** - Select the variable field you want to use. The contents of this list depend on which variable you selected in the **Variable** property.

Field 

Type

Variable item

Available from

Variables Editor, with complex variables only. For more information about variables and the Variables Editor, see Working with Variables.

Purpose

A Complex Variable is a variable that has several attributes, called *fields*. Complex variables make it possible to use a single variable to pass along multiple values. To be valid, a complex variable must have at least one **Field** item assigned to it.

Behavior

In a sense, a **Field** item acts like a simple variable within a complex variable. In other words, each **Field** item contains a single variable value.

When you create a complex variable, Dialog Designer automatically assigns a temporary name to it and adds one **Field** item. You can rename both the variable and the field. You can also add as many additional **Field** items as you want.

In some cases, Dialog Designer creates complex variables automatically when you use certain nodes or items in the call flow. For complex variables that Dialog Designer creates automatically, the fields are also created automatically. As a general rule, you cannot add **Field** items to these variables. For example, when you place a Bridged Transfer item in the call flow, Dialog Designer automatically creates a complex variable with the same name as the item. This complex variable automatically has four fields created and assigned to it: **duration**, **inputmode**, **utterance**, and **value**.

Properties

- **Name** - Enter in this field the name you want to identify the variable field.

  Note:
  This name must follow conventions for naming Java components as described in Naming Java Components.

- **Value** - Enter in this field the default value you want to assign to the variable field, if any. If you leave this field blank, the default value is undefined.

Flush Prompts

Type

Form item

Available from

- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Form Node

Purpose

The **Flush Prompts** item causes queued output to be played before transitioning to the next node.

Behavior

When executed, the **Flush Prompts** item directs the voice browser to place all queued output immediately.

Only one **Flush Prompts** item can be added in any one node. An example of where the **Flush Prompts** is useful is when doing a CTI transfer. When doing a transfer one would typically play a prompt “please wait while I transfer your call”. If a caller does not hear a particular prompt, a Flush Prompt item may need to be added just before the prompt item that is not playing. This forces any queued prompts to play before moving on.

It does not matter where the **Flush Prompts** item appears in the subflow of a node.

Properties

- None

Get VDU Fields

Type

IC Connector item

Available From

Data Node (IC section)

Purpose

Get VDU Fields allows you to get up to 20 vdu variables in one round trip to the VOX.

Behavior

When this node is invoked, the data will be retrieved from the VOX and placed into the **vdu_cache** complex variable into the field names that match the names of the vdu fields indicated. You can then access each variable out of the cache.

Properties

- **vdu_field1 - vdu_field20** - Indicates the name of the vdu field to retrieve. You must first add the field to the vdu complex variable.

Goto 

Type

Form item

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Menu Node
- Form Node
- Servlet Node

Purpose

The **Goto** item makes it possible to redirect a call flow to a specific node in response to an event.

For example, suppose a caller is having trouble making a response that the system can match. You might, in this case, want to redirect the caller to a live attendant. You can set the **No Match** event handler to go to a Blind Transfer node, which transfers the caller to the attendant.

Behavior

When the system catches an event to which the **Goto** item is assigned, the application directs the call to the node specified in the **Goto** item.

With the exception of the Servlet node, the **Goto** item cannot be used independently. It is always used as a sub-item of an event handler.

Note:
In the Servlet node, the **Goto** item is used independently.

The **Goto** item cannot be used with same event handler as a **Throw** item.

Properties

- **Goto Form** - Select the node to which you want to direct the call when the specified event is caught.
- **Threshold** - A variable that must have a positive integer value.

  The **Threshold** setting determines how many times the specified event must be caught before application redirects the call flow to the **Goto Form**. Until this threshold is reached, the system plays the prompt assigned to the event handler, if any, and then waits for another response from the caller.

Grammar

Type

Form item

Available from

Any node in which a spoken response from a caller is possible:

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Form Node

Purpose

An integral part of self-service speech applications is the ability to collect responses from callers and use them to direct the call flow. These caller responses can be in the form of DTMF key presses or spoken commands.

Grammars are used at those points in a speech application where you anticipate a spoken or DTMF response from a caller. Grammars define what words, phrases, or key presses the system can recognize and use as input. **Grammar** items make it possible to select from among all the grammars defined for an application, to use at specific places in the application.

Note:
In the **AppRoot** node, a grammar can be used only in conjunction with a **Link** item. This grammar is active on a global basis. That is, the system responds to any utterance that matches this grammar, no matter where the caller is in the call flow. See Link and Setting Global Application Properties: AppRoot Node.

For more information about grammars, see Working with Grammars.

Behavior

Grammar items cannot be used independently in nodes. Grammar items must always be used as sub-items of:

- Bridged Transfer items
- Input items
- Link items
- Record items

In short, you can use grammars only in places where it makes sense to collect and use spoken or DTMF input from callers. You must create the grammar file itself before you can use it in a node.

When a call reaches a node in which there is an active grammar, the system monitors input from the caller. Whenever a caller utterance or key press matches an item in the grammar, the system processes the input according to how the node has been set to respond to that input.

For more detailed information about creating and using grammars, see Working with Grammars.

Properties

- **Name** - Select the grammar you want to use.

Note:
You can select only one grammar per **Grammar** item. If you want to use multiple grammars at any point, you must use one **Grammar** item for each grammar you want to use. For example, if you want to offer callers the option to respond either with a key press of **1** or by saying the word “one,” you can use two grammars: one to recognize the DTMF input and the other to recognize the spoken response.

- **Weight** - (Default: 1.0) Optional. Enter in this field any floating point number of the format *n*, *n.*, *.n*, or *n.n*, where *n* represents any length sequence of digits. A weight of 1.0 is considered neutral. That is, the grammar with a weight of 1.0 does not bias the recognition one way or the other. A weight of greater than 1.0 biases the grammar in a positive way. A weight of less than 1.0 biases the grammar in a negative way.

  The exact application of weights depends on the ASR engine being used. Therefore, a weight that works well on a particular platform might produce different results on other platforms.

  Weights have no effect on DTMF grammars.

If 

Type

Condition item

Available from

Any level tab in the Prompt File Editor

Data Node in the Call Flow Editor

Purpose

The **Condition** item makes it possible to select and play different prompt or data flow segments based on the condition being tested. It works in conjunction with **If** and **Else** items to test conditions and return a result. The **If** item makes it possible to compare the value of a selected variable, the **Left Variable** or **Left Variable Field** with:

- A condition
- A constant
- The value of another variable or variable field

Then, based on the result of the comparison, the system responds according to what is assigned to the **If** item, or continues on.

Behavior

The behavior of this item depends largely on which conditional **Operator** is selected. In general, when the value of the **Left Variable** is compared and satisfies the condition, then the application plays any prompt segments assigned to the **If** item. For more information about conditional operators and how to use them, see Conditional Operators.

You can use multiple **If** items with a single **Condition** item. If you do so, then the application tests for each **If** item in turn, in the order they appear under the **Condition** item. This process continues until the conditions for one **If** item are met, or there are no more conditions to test for, that is, there are no more **If** items to test.

Properties

- **Left Variable** - Select the variable you want to use to make the comparison.

  If the variable you select is a simple variable, this is the variable to use for the comparison. If the variable you select is a complex variable, you must select a variable field from the **Left Variable Field** list to use for the comparison.

  Note:
  For more information about simple and complex variables, see Working with Variables.

- **Left Variable Field** - If the variable you selected in the **Left Variable** field is a complex variable, select the field that you want to use to make the comparison. If the variable you selected in the **Left Variable** field is a simple variable, this field is inactive and no choices are available.

- **Compare** - Select the conditional operator you want to use to make the comparison. For more information about conditional operators and how to use them, see Conditional Operators.

- **Value** - Use this field when you want to compare the value of the **Left Variable** or **Left Variable Field** with a literal, or constant, value. Enter in this field the literal value you want to compare the **Left Variable** or **Left Variable Field** with.

  For example, suppose you want to compare the account balance to see if it has a minimum value of at least $2000, and if it is, you want to play this prompt instead of another. To do this, you enter $2000 in this field and the appropriate conditional operator in the **Compare** field.

  If you use the **Value** field, you cannot use the **Right Variable** field.

- **Right Variable** - Select the variable you want to compare with the Left Variable or Left Variable Field.

  If the variable you select is a simple variable, this is the variable to use for the comparison. If the variable you select is a complex variable, you must select a variable field from the **Right Variable Field** list to use for the comparison.

  Note:
  For more information about simple and complex variables, see Working with Variables.

  If you use the **Right Variable** field, you cannot use the **Value** field.

- **Right Variable Field** - If the variable you selected in the **Right Variable** field is a complex variable, select the field that you want to use to make the comparison. If the variable you selected in the **Right Variable** field is a simple variable, this field is inactive and no choices are available.

Input 

Type

Form item

Available from

- Announce Node
- Prompt and Collect Node
- Form Node

Note:
The **Input** item cannot be used in the same node as the Blind Transfer item, the Bridged Transfer item, or the Record item. So, although the **Input** item appears on palettes for other nodes, such as the Blind Transfer node, the Bridged Transfer node, and the Record node, it is not actually available for use in those nodes without destroying their intended functionality. It is intended primarily for use in the Prompt and Collect node.

Purpose

The **Input** item is used to collect and process responses from callers. The responses can be in the form either of DTMF key presses or of spoken responses. Usually, the collected input is then used in some way to direct the flow of the call. The collected input can also be used as data that can be used by the application in a variety of ways.

Note:
The **Input** item in Dialog Designer corresponds with the *field* within a *form* in the VoiceXML Version 2.0 W3C Recommendation. See the W3C VoiceXML 2.0 Recommendation.

Behavior

In Dialog Designer, the primary means of collecting and using inputs from callers is through the Prompt and Collect node. In a typical scenario, the system plays a prompt to cue the caller what type of response is expected, and then it waits for the caller to respond. The **Input** item is the node element that is responsible for accepting and processing the caller response. When it receives an input from a caller, it submits that input to the ASR engine and then waits for a return result from the ASR engine.

The one essential sub-item of the **Input** item is the **Grammar** item. Grammars determine what type of response the system is listening for and can recognize. DTMF grammars determine what touchtone key press or combination of key presses can be recognized and processed. Voice grammars determine what words or phrases can be recognized and processed. You can use both DTMF grammars and voice grammars with a single **Input** item. For more information about creating and using grammars, see Working with Grammars.

In addition to the **Grammar** sub-item, the **Input** item can also take any of the standard event handlers. The Prompt and Collect node automatically includes the **No Input** and **No Match** event handlers, but you can add any others you want.

When you place an **Input** item in the call flow, Dialog Designer automatically creates a complex variable with the same name.

Note:
Because the Prompt and Collect node, by design, includes an **Input** item, when you place a Prompt and Collect node in the call flow, Dialog Designer automatically creates a complex variable with the same name as the **Input** item.

This complex variable contains the following fields, which are populated upon completion of the node:

- **confidence** - A number between 0.0 and 1.0 that expresses the degree of confidence the system has that it has correctly recognized the caller utterance. The number 0.0 indicates minimum confidence, and the number 1.0 indicates maximum confidence. The specific interpretation of this number depends on the ASR platform.
- **utterance** - A text (raw string) summary of what the caller said or keyed in, as recognized by the ASR engine. In the case of a DTMF response, this is the digit sequence of keys that the caller pressed.
- **inputmode** - The mode in which the caller input was provided, either voice or DTMF.
- **interpretation** - The contents of the interpretation tag, as provided by the ASR engine. In other words, this is the result of how the ASR engine recognized and mapped the utterance. If the ASR engine is unable to provide an interpretation, this field remains undefined.
- **value** - If the ASR engine returned an interpretation, this field takes on the same value as the **interpretation** field. If not, this field takes on the value of the **utterance** field.

Once the ASR engine returns a recognition result and populates these Input variable fields, you can use the outcome to further direct the call.

Properties

- **Name** - Enter the name you want to assign to the **Input** item and its corresponding variable.

  If you change the name of this item in the **Properties** view, Dialog Designer automatically changes the name of the associated variable as well. If the **bind name to node** attribute is set, then renaming the parent node will rename this field. (This applies to all implicit variable items.)

  Note:
  This name must follow conventions for naming Java components as described in Naming Java Components.

- **N-Best** - Enter the maximum number of recognition results to allow in the case of multiple recognition results from the ASR engine.

  The default for this property is **1**.

- **Modal** - Select your choice from the drop-down list. Options are:
  - **True** – The system ignores all other grammars that might be active at this point, except the grammar associated with this **Input** item.

    Setting this property to **True** can be helpful when concerned about a a false match that might redirect the call flow to another part of the application when you do not want it to. In a case like this, you want to make sure that a caller response matches a grammar associated with the **Input** item, if at all possible.
  - **False** (default) – The system recognizes all active grammars. If the system matches a caller response with an item in a grammar other than the grammars associated with this **Input** item, control passes to the item associated with the other grammar, and any data collected within the current node is lost.
  - **Record Utterance** – (false/true). When true, additional fields are added to the variable that store the recording of the caller's utterance, the size of the recording (in bytes) and the duration of the recording (in milliseconds).
  - **Collect Mark Data** – (false/true). When true, additional fields are added to the variable that store the name of the last **Mark** encountered by the VoiceXML platform as well as the time elapsed since the mark was processed.
  - **Bind Name to Node** – (true/false). When true, the name of the variable item is bound to the name of the node. When the parent node is renamed, the name of the variable item is renamed as well; keeping the name of the variable in sync with the node name.

Input Parameter 

Type

Application item

Available from

AppRoot node only (see Setting Global Application Properties: AppRoot Node)

Purpose

When using a modular approach to application design, it is often necessary to pass variable or constant values from one module to another. The **Input Parameter** item makes it possible for a speech project used as a module node to accept such input from another speech project. For more information about creating and using application modules, see Module Nodes.

Behavior

Use the **Input Parameter** item when you want to define values to be passed in to a speech project that you want to use as a module for other speech projects. To pass values in to that module, you must place into the **AppRoot** node one **Input Parameter** item for each value that you want to pass in to the module.

For each **Input Parameter** that you place in the **AppRoot** node, Dialog Designer automatically creates a simple variable with the same name. You can use these variables the same as you would use any other variable. If a default value is not assigned to the **Input Parameter** item, Dialog Designer generates an error. When you test the module project, the Avaya Application Simulator (AAS) displays these **Input Parameter** variables in the **Input Parameters** display pane of the **Call** tab. For more information about testing projects, see Application Testing by Simulation.

When you are satisfied that your module works the way you want it to work, generate and save it. You must then deploy it as a Dialog Designer module. For more information about this and the procedure to deploy it, see Deploying Projects as Dialog Designer Modules.

Later, when you use this speech project as a module in another project, Dialog Designer automatically populates the module node with one **Module Input** item for each **Input Parameter** item in the module. See Module Input.

Properties

- **Name** - Enter the name you want to use to identify the value being passed in and its corresponding variable.

  Dialog Designer assigns a default name automatically when you place the **Input Parameter** item in the node, and you can leave it as is or rename it.

  Note:
  This name must follow conventions for naming Java components as described in Naming Java Components.

- **Description** - (Optional) Enter in this field a brief description of what the input parameter value is used for or expected to contain. Although this is optional, Avaya recommends that you write a good description. The reason is that it can be helpful as a reminder of what the input parameter is for, later on when you or others use the module in another speech project.
- **Default Value** - Enter in this field the default value you want the variable to have. If you do not enter a value in this field, Dialog Designer generates an error when you save or generate the project.

Invoke Workflow 

Type

IC connector item

Available from

Data Node only

Purpose

The **Invoke Workflow** item makes it possible to invoke and use a workflow on an Avaya Interaction Center (IC) system without exiting the speech application on the IVR system. With the **Invoke Workflow** item, you can pass variable values to the workflow on the IC system. The workflow on the IC system can process those variable values and then return the results to the speech application on the IVR system.

Behavior

Before you can use this or any other IC item, you must enable IC for Dialog Designer. For more information about the ability of Dialog Designer to interact with IC, and to enable IC in your Dialog Designer applications, see About the IC Connector.

When the speech application encounters the **Invoke Workflow** item, the application:

- Sends a command to the IC system to invoke the workflow identified in the **Workflow Name** property.

  For more information about this property, see the “Properties” section that follows.

- Passes the values of any variables selected in the input variables and input variable fields *in order* to the IC workflow.

Note:
The fact that these values are passed in the order they are listed in the Properties view means that you must know the order in which variable values are expected in the workflow.

- Waits for the IC workflow to finish and return any results.

  Note that the amount of time that the Dialog Designer application waits for a return from the IC system is determined by the **Max Timeout in ms** property. See the description of this property in the “Properties” section that follows.

  The **Return Immediately** property can be set to true or false (default) to have the invoked call return without waiting for the workflow to finish. If this flag is set, you won't be able to enter output variables or a **max Timeout** value.

Note:
The **vox.tr1** call allows you to perform a send and forget when invoking a workflow. The IC Connector will not wait for the results of the running workflow and the application may continue executing. Enable this feature by setting the **Return Immediately** property in the IC Invoke Workflow data node to true.

- Receives the results passed back to the application by the IC workflow and assigns them to the designated output variables and output variable fields.

Important:
When passing variable values to and from the IC system, keep in mind that, whereas the IC system allows dots (**.**), pluses (**+**), stars (*****), and marks(**!**) in variable names, Dialog Designer, being Java-based, does not. So, for example, a variable name of **account.number** is perfectly valid in the IC system, but it violates Java naming conventions, which do not allow the dot in a name.

Therefore, when creating variables to pass values to known variables on the IC system or to receive variable values from known variables on the IC system, you must use the following naming convention:

Replace the dot (**.**) with the following: __**DOT**__

Replace the plus (**+**) with the following: __**PLUS**__

Replace the star (*****) with the following: __**STAR**__

Replace the mark (**!**) with the following: __**MARK**__

This mnemonic replacement consists of two underscore (_) characters, followed by the word **DOT**, **PLUS**, **STAR**, or **MARK** (all uppercase), as applicable, followed by two more underscore characters.

At run time, the Dialog Designer application translates this mnemonic replacement for the dot, whether for variable values being passed to the IC system or received from the IC system.

Tip:
There are two ways that variable values can be passed to and from the IC system: One is to use the **Invoke Workflow** item, and the other is to create custom variable fields in the **vdu** variable and assign values to them. While the second method is more roundabout, it can be useful if you need to pass more than four variable values at a time. This is because the **Invoke Workflow** item has a limit of four output and four input variable values. By creating and assigning custom variable fields in the **vdu** variable *before* the **Invoke Workflow** item is initiated, you can send additional variable values to the IC system for use with the **Invoke Workflow** item.

Properties

- **Workflow Name** - Enter the name of the workflow you want to invoke on the IC system.

  This name must match *exactly* the name of a workflow on the IC system.

- **Max Timeout in ms** - Enter the amount of time in milliseconds to allow the IC system to invoke the workflow, complete its process, and return any results.

  If the IC system takes longer than this amount of time to return a response, the IVR system throws a run-time error and responds appropriately.

Tip:

This setting requires you to know the normal amount of time it takes the IC system to complete the workflow you are invoking. If, for example, you are invoking a workflow that normally takes from 4 to 7 seconds to process a request, and you have set this timeout to **5000** milliseconds (5 seconds), there is a good chance that the IVR application will time out before the request can be processed. As a general rule, Avaya recommends that you set this field to approximately 5 seconds more than the normal processing time that the workflow requires. In this example, that means that you should set this field to **12000**.

- **Input Variable1** through **4** - For each input variable, select the variable whose value you want to pass to the IC workflow. You must assign the variables in the order the IC workflow expects to receive them.

  If a selected variable is a complex variable, you must also select a corresponding **Input Variable Field** for that variable.

- **Input Variable Field1** through **4** - If a given **Input Variable** is a complex variable, select the corresponding variable field whose value you want to pass to the IC workflow.

- **Output Variable1** through **4** - For each output variable, select the variable to which you want the IC workflow to return the results of processing.

Note:

The IC system creates output values using *name-value* pairs. Dialog Designer applications can retrieve those name-value pairs only if it knows the names. For this reason, the name of the variable or variable field to which you want to assign the return from the IC system must match *exactly* the name of one of the names in these name-value pairs.

If a selected variable is a complex variable, you must also select a corresponding **Output Variable Field** for that variable.

- **Output Variable Field1** through **4** - If a given **Output Variable** is a complex variable, select the corresponding variable field to which you want the IC workflow to return the results of processing.

Join Condition

Type

Data item (Database Operation Condition)

Available From

Data Node

Database Operation Editor Predicate Tab

Purpose

For setting up a condition that specifies a join between two tables. You can use the combination of the Simple and Join condition to create a complex search criteria.

Behavior

When adding a Join item, two Column item and a Comparison item are automatically created inside the Join item. You just need to specify the properties of the three nested items, and you have a join condition.

Properties

None

Link 

Type

Form item

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Menu Node
- Form Node

Purpose

The **Link** item is, essentially, a grammar-activated **Goto** item. The **Link** item makes it possible to redirect the call flow based on a particular spoken word or phrase or a combination of DTMF key presses. This functionality can be helpful, for example, when you want to allow callers to go to a live operator at any point in the call flow. In this case, you might place a **Link** item in the **AppRoot** node with grammars to recognize both the word “operator” and a key press of **0**. Either response by a caller would redirect the call flow to a call transfer node to dial the extension for the live attendant.

Behavior

The **Link** item has built-in DTMF recognition capability. This means that you do not have to use a DTMF grammar for the application to recognize a DTMF response. If you want the **Link** item to recognize spoken responses, however, you must attach an ASR **Grammar** item to the **Link** item.

You must use either one **Goto** item or one **Throw** item with the **Link** item. You can only use one or the other, not both.

If you use a **Goto** item, when the **Link** is activated, the application passes control to the node designated by the **Goto** item. See Goto.

If you use a **Throw** item, when the **Link** is activated, the application throws the event designated by the **Throw** item. See Throw.

Properties

- **Name** - Enter in this field the name you want to use to identify the **Link** item.

  For **Link** items that use a **Goto** item, this name appears as a connection point in the call flow.

  Note:
  This name must follow conventions for naming Java components as described in Naming Java Components.

- **DTMF** - Enter in this field the DTMF key press or key press combination that you want to use to activate the **Link** item.

  Note:
  You can, if you prefer, use a DTMF grammar for DTMF recognition. The advantage of using a DTMF grammar is that you can use multiple DTMF key press combinations to activate the **Link** item.

Mark 

Places a marker into a text/tag sequence. An SSML processor must either allow the VoiceXML interpreter to retrieve or must inform the interpreter when a <mark> is executed during audio output.

Type

SSML item

Available from

Any level tab in the Prompt File Editor.

See Using the Prompt File Editor.

Purpose

The **Mark** item is a Synthesized Speech Markup Language (SSML) element that is used to place markers in a prompt that plays TTS. The markers can be used for detecting barge-in during prompt playback as well as tracking how much of a prompt the caller heard based on mark names and times.

Behavior

Marks are processed by the TTS engine as they are encountered during the speech synthesis. When a mark is encountered, the mark name is sent to the VoiceXML platform which tracks the name of the last mark encountered and the time since it was processed.

Module Input 

Type

Application item

Available from

Module Nodes only

Purpose

When using a modular approach to application design, it is often necessary to pass variable or constant values from one module to another. The **Module Input** item makes it possible for a speech project to pass values into another speech project used as a module. For more information about creating and using application modules, see Module Nodes.

Behavior

When you place a module node in the call flow, Dialog Designer automatically populates it with one **Module Input** item for each input that the module can accept. If the module was created with Dialog Designer, this means that there is one **Module Input** item for each **Input Parameter** item in the speech project being used as the module. See Input Parameter.

If the module was not created with Dialog Designer, you can specify inputs to the module when you import the module into Dialog Designer. Then, when you use the module in a Dialog Designer speech application, Dialog Designer automatically populates it with the required **Module Input** items, much like it does with Dialog Designer-created modules.

Note:
Currently, Dialog Designer supports only Nuance OSDMs for import into Dialog Designer applications. Dialog Designer supports the following OSDM versions: Address OSDM 2.0.3, Core OSDM 2.0.4, and Name OSDM 2.0.1.

To use these items in the speech project, you must assign a value to the **Module Input** item in the module node. This value can be either a variable value or a constant, but if you do not assign a value in the module node properties, no value is passed to the module node at run time.

Properties

- **Name** - (Display only) This name is the name of the item in the module that can accept input. If the module was created with Dialog Designer, this is the name assigned to the **Input Parameter** item. You cannot change it here.
- **Description** - (Display only) This field is the description entered into this field when the module project was created.
- **Default Value** - (Display only) This field is the default value entered into this field when the module project was created.
- **Variable** - Select the variable whose value you want to pass in to the module. If the variable you select is a complex variable, you must also select a **Variable Field**.

  If you enter a **Constant** value, you cannot also use this field.
- **Variable Field** - If the variable you selected in the **Variable** field is a complex variable, you must select from this drop-down list the field whose value you want to pass in to the module. If the variable you selected in the **Variable** field is a simple variable, this field is blank.
- **Constant** - Enter in this field the exact string or value you want to pass in to the module.

  If you use this field, you cannot also use the **Variable** field.

  Important:
  If you use the **Constant** property, be careful that the form and format you enter match the form and format expected in the module that is receiving it.

Module Output 

Type

Application item

Available from

Module Nodes

Purpose

When using a modular approach to application design, it is often necessary to pass variable values from one module to another. The **Module Output** item makes it possible for a speech project used as a module node to pass such data back to the parent speech project.

Behavior

When you place a module node in the call flow, Dialog Designer automatically populates it with one **Module Output** item for each output value that the module passes to the parent application. If the module was created with Dialog Designer, this means that there is one **Module Output** item for each **Output Parameter** item in the speech project being used as the module. See Output Parameter.

If the module was not created with Dialog Designer, you can specify outputs from the module when you import the module into Dialog Designer. Then, when you use the module in a Dialog Designer speech application, Dialog Designer automatically populates it with the required **Module Output** items, much like it does with Dialog Designer-created modules.

Note:
Currently, Dialog Designer supports only Nuance OSDMs for import into Dialog Designer applications. Dialog Designer supports the following OSDM versions: Address OSDM 2.0.3, Core OSDM 2.0.4, and Name OSDM 2.0.1.

At the same time, Dialog Designer automatically creates:

- A complex variable with the same name as the module node.
- A field for that variable with the same name as the name of the **Module Output** item in the module node.

You can use the returns to this variable and variable field the same as you would use any other variable and variable field.

Properties

- **Name** - (Display only) This field is the name assigned to the **Module Output** item when the module was named. If the module was created with Dialog Designer, this name is the same as the name of the **Output Parameter** item in the speech project being used as the module. You cannot change it here.
- **Description** - (Display only) This field is the description entered into this field when the module project was created.
- **Default Value** - (Display only) This field displays the value of the variable assigned to the **Variable** property of the **Output Parameter** item in the module that is passing the variable information to the parent application.

Next

Type

Form item

Available from

- Announce Node

- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Form Node
- Data Node
- Tracking Node

Purpose

The **Next** item determines or indicates what is to be the next node in the call flow.

Behavior

When executed, the **Next** item directs the call flow to the node indicated in the **Next Form** field.

You can have only one **Next** item in any one node. Most nodes that require the use of a **Next** item include it by default. This includes all the nodes listed in the previous section except the Form node.

No matter where the **Next** item appears in the subflow of a node, it is always executed last.

Properties

- **Next Form** - Select the node to which you want the call flow to proceed when the current node is finished executing.

Note:
You can also populate this field by drawing a **Connection** line in the main workspace of the Call Flow Editor. See Using the Connection Option.

No Input 

Type

Event handler

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node

- Record Node
- Menu Node
- Form Node

Purpose

When a caller does not respond to a request for input within a specified period of time, the system throws a No Input event. The **No Input** event handler makes it possible to specify how the system responds when these No Input events are thrown.

Behavior

Within the scope defined for it, the **No Input** event handler waits for a No Input event to be thrown. When such an event is thrown, the **No Input** event handler responds according to the options you use with it. See the next section, “Properties.”

The **No Input** event handler responds to a No Input event anywhere within the scope of the handler. For example, if you place the **No Input** event handler in the **AppRoot** node, it acts as a global event handler for No Input events. It can catch No Input events no matter where in the application the events are thrown. If you place the **No Input** event handler in another node, however, it can catch a No Input event only if it is thrown while in that node. If you place the **No Input** event handler underneath a node item, it can catch the No Input event only if it is thrown within that node item.

A **No Input** event handler at the node item level overrides a **No Input** event handler at the node level. Likewise, a **No Input** event handler at the node level overrides a **No Input** event handler at the **AppRoot**, or global, level.

Note:
This event handler is one of the default event handlers defined by the W3C VoiceXML 2.0 Recommendation. There it is identified as a *noinput* event handler.

Properties

The **No Input** event handler has no inherent properties, but you must use one or more of the following options with it:

- **Prompt** item - Plays the designated prompt to the caller before the application continues.
- **Goto** item - Directs the application to the designated node.
- **Throw** item - Throws an event that you choose. Select a user-defined event that you have created.

  By throwing an event within an event handler, you can transfer control to a different part of the application, for example, to the **AppRoot** node. For more information about throwing events and handling events, see Throw or Working with Event Types.

No Match 

Type

Event handler

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Menu Node
- Form Node

Purpose

When a caller does not respond to a request for input with a response that matches any that the system is set to recognize, the system throws a No Match event. The **No Match** event handler makes it possible to specify how the system responds when these No Match events are thrown.

Behavior

Within the scope defined for it, the **No Match** event handler waits for a No Match event to be thrown. When such an event is thrown, the **No Match** event handler responds according to the options you use with it. See the next section, “Properties.”

The **No Match** event handler responds to a No Match event anywhere within the scope of the handler. For example, if you place the **No Match** event handler in the **AppRoot** node, it acts as a global event handler for No Match events. It can catch No Match events no matter where in the application the events are thrown. If you place the **No Match** event handler in another node, however, it can catch a No Match event only if it is thrown while in that node. If you place the **No Match** event handler underneath a node item, it can catch the No Match event only if it is thrown within that node item.

A **No Match** event handler at the node item level overrides a **No Match** event handler at the node level. Likewise, a **No Match** event handler at the node level overrides a **No Match** event handler at the **AppRoot**, or global, level.

Note:
This event handler is one of the default event handlers defined by the W3C VoiceXML 2.0 Recommendation. There it is identified as a *nomatch* event handler.

Properties

The **No Match** event handler has no inherent properties, but you must use one or more of the following options with it:

- **Prompt** item - Plays the designated prompt to the caller before the application continues.
- **Goto** item - Directs the application to the designated node.
- **Throw** item - Throws an event that you choose. Select a user-defined event that you have created.

  By throwing an event within an event handler, you can transfer control to a different part of the application, for example, to the **AppRoot** node. For more information about throwing events and handling events, see Throw or Working with Event Types.

Object 

Type

Application item

Available from

- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Form Node

Purpose

The **Object** item makes it possible to extend the functionality of VoiceXML beyond its normal capabilities. For example, the VoiceXML standard does not support conference calling. However, if your system supports conference calling, you can write a platform-specific script or application that can be used to take advantage of your system's capabilities. You then use this script or application using the **Object** item.

Behavior

The **Object** item takes the values of the properties specified in the following section and uses them to implement and execute the object specified.

If you need to pass data to the object from the application, use the **Object Input** item. See Object Input.

If the object returns data that you want to use in the application, use the **Object Output** item. See Object Output.

For more information about using objects in VoiceXML, see “object element” in the W3C VoiceXML 2.0 Recommendation.

Properties

Of the following properties for the Object item, only the Class ID property is required for all objects. The need to use other properties depends on what object you are calling.

- **Name** - Enter the name to identify the object in the call flow.
- **Class ID** - Enter the URI for the object you want to execute.

  This URI may be either an absolute or a relative URI. If it is a relative URI, it is interpreted relative to the **Codebase** property.
- **Data** - Enter the URI that specifies the location of the data for the object.

  This URI may be either an absolute or a relative URI. If it is a relative URI, it is interpreted relative to the **Codebase** property.
- **Type** - Enter the content type of the data specified in the **Data** property.
- **Archive** - Enter a space-separated list of URIs for archives containing resources relevant to the object.

  These resources can include the resources specified by the **Class ID** and **Data** properties.

  Any relative URIs are interpreted relative to the **Codebase** property.
- **Codebase** - Enter the base path to use when resolving relative URIs specified by the **Class ID**, **Data**, and **Archive** properties.

  The default for this property is the URI of the current document.
- **Codetype** - Enter the content type of the data the system expects when it downloads the object specified by the **Class ID** property.

  If you leave this field blank, the system defaults to the content type specified in the **Type** property.

Object Input 

Type

Application item

Available from

- Announce Node
- Prompt and Collect Node

- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Form Node

Purpose

When you use an Object item to extend the functionality of VoiceXML, you might need to pass variable values from the application to the object. The **Object Input** item makes it possible to pass these values to the object.

Behavior

The **Object Input** item takes the value assigned and passes it to the object when the object is implemented. For information about the values that can be passed, see the following section, “Properties.”

The **Object Input** item must be associated with an **Object** item. For more information about **Object** items, see Object.

Properties

- **Name** - Enter the name you want to identify the **Object Output** item in the node subflow.
- **Constant Value** - Enter the constant value you want to pass to the object.
- **Expression Value** - Enter the ECMAscript expression you want to pass to the object.

  CAUTION:
  Because the expected content for this property is an ECMAscript expression, Dialog Designer applications pass the value of this property directly to the Avaya Voice Browser (AVB). Avaya recommends that you use this property with caution.
- **Variable Value** - Select the variable whose value you want pass to the object.

  It the selected variable is a complex variable, you must also select a **Variable Field**.
- **Variable Field Value** - If the variable selected in the **Variable** field is a complex variable, select the associated variable field whose value you want to pass to the object.

Note:
If the variable selected in the **Variable** field is a simple variable, this list is not populated.

Object Output 

Type

Application item

Available from

- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Form Node

Purpose

When you use an Object item to extend the functionality of VoiceXML, you might need to pass variable values back from the object to the application. The **Object Output** item makes it possible to pass these values back to the application.

Behavior

The **Object Output** item must be attached to an **Object** item. For more information about **Object** items, see Object.

When you attach an **Object Output** item to an **Object** item, Dialog Designer automatically creates:

- A complex variable with the same name as the associated **Object** item.
- A field for that variable with the same name as the **Object Output** item.

Tip:
If you want to assign a default value to this variable field, do so in the Variable Editor. See Working with Variables.

You can attach multiple **Object Output** items to a single **Object** item. Dialog Designer creates a new variable field for each **Object Output** item you attach to the **Object** item.

Tip:
After you attach the **Object Output** item to the **Object**, the new variable and variable field do not appear in the Variable Editor until you save the project. After you save the project, you must close the Variable Editor, if it is open, and reopen it.

You can use these variables and variable fields the same as you would use any other variable and variable field.

Properties

- **Name** - Enter the name that identifies the **Object Output** item in the node subflow and its corresponding variable field.

Note:
The name you enter in this property field must match the name of the object variable being returned from the module.

On Disconnect 

Type

Event handler

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Menu Node
- Form Node

Purpose

When a caller hangs up or otherwise gets disconnected from the system before the application is done executing, the system throws an On Disconnect event. The **On Disconnect** event handler makes it possible to specify how the system responds when these On Disconnect events are thrown.

Behavior

Within the scope defined for it, the **On Disconnect** event handler waits for an On Disconnect event to be thrown. When such an event is thrown, the **On Disconnect** event handler responds according to the options you use with it. See the next section, “Properties.”

The **On Disconnect** event handler responds to an On Disconnect event anywhere within the scope of the handler. For example, if you place the **On Disconnect** event handler in the **AppRoot** node, it acts as a global event handler for On Disconnect events. It can catch On Disconnect events no matter where in the application the events are thrown. If you place the **On Disconnect** event handler in another node, however, it can catch an On Disconnect event only if it is thrown while in that node. If you place the **On Disconnect** event handler underneath a node item, it can catch the On Disconnect event only if it is thrown within that node item.

An **On Disconnect** event handler at the node item level overrides an **On Disconnect** event handler at the node level. Likewise, an **On Disconnect** event handler at the node level overrides an **On Disconnect** event handler at the **AppRoot**, or global, level.

Note:
This event handler is one of the default event handlers defined by the W3C VoiceXML 2.0 Recommendation. There it is identified as a *connect.diconnect* event handler.

Properties

The **On Disconnect** event handler has no inherent properties, but you must use one or more of the following options with it:

- **Goto** item - Directs the application to the designated node.
- **Throw** item - Throws an event that you choose. Select a user-defined event that you have created.

  By throwing an event within an event handler, you can transfer control to a different part of the application, for example, to the **AppRoot** node. For more information about throwing events and handling events, see Throw or Working with Event Types.

Operation 

Type

Data item

Available from

Data Node only

Purpose

The **Operation** item, labeled **Undefined Data Operation** in the subflow, makes it possible to manipulate the values of variables in a variety of ways.

Behavior

The **Operation** item takes the value of a variable or variable field and manipulates it according to the selection of the **Type** property. When you select the type, the name of the **Operation** item changes to match the type. The Properties view also changes to reflect the additional properties associated with that type.

The specific behavior of each defined type is described in the table that follows.

Properties

The properties of the **Operation** item depend on which **Type** you assign to the item. Only one property is common to all variants of the **Operation** item: the **Type** property. What is selected for this property determines what other properties are available. The following table lists all the available types, along with the other properties associated with each type:

Operation Item Property Types

Type	Associated properties	Comments/Remarks
Assign - Takes the value of the source and assigns it to the destination. If no source value is specified, that is, all three **Source...** fields are left blank, the system assigns the value of an empty string to the destination variable.	**Destination Variable (and Field)**	The variable that the source value is assigned to . If the destination is a complex variable, a field must be selected.
	Source Variable (and Field)	Select the variable to assign to the destination. If this variable is a complex variable, a **Source Variable Field** must be selected whose value should be assigned to the destination. If this property is used, the **Source Constant** property cannot be used.
	Source Constant	Enter the constant value you want to assign to the destination variable. If this property is used, the **Source Variable** property cannot be used.

Operation Item Property Types (continued)

Type	Associated properties	Comments/Remarks
Concatenate - Concatenates to, or adds to the end of the value of, the destination variable or variable field. Usually, the variable or variable field value is of type String. Because this is a String-based operation, you can concatenate numbers, but they are then string-based numbers. For example, if you concatenate a **3** to a **4**, you end up with the value **34**.	**Destination Variable (and Field)**	Select the variable to which the source value is to be concatenated. If the destination is a complex variable, a field must be selected..
	Source Variable (and Field)	Select the variable (and field if Source Variable is a complex variable) whose value is to be concatenated to the destination. If the source is a complex variable, a field must be selected.
	Source Constant	Enter the constant value you want to concatenate to the destination variable. If this property is used, the **Source Variable** property cannot be used.
Length - Returns the number of characters in a source variable string, and assigns that number to the destination variable.	**Destination Variable**	Select the variable to which the source length is to be assigned. If this variable is a complex variable, a **Destination Variable Field** must also be selected.
	Destination Variable Field	If the **Destination Variable** is a complex variable, select the field to which the source length is to be assigned.
	Source Variable	Select the variable whose length you want to assign to the destination variable. If this variable is a complex variable, you must also select a **Source Variable Field**.
	Source Variable Field	If the **Source Variable** is a complex variable, select the field whose length you want to assign to the destination variable.

Operation Item Property Types (continued)

Type	Associated properties	Comments/Remarks
Size - If the source variable is a collection, returns the size of the collection. That is, this type returns the number of items in the collection. If the source variable is not a collection, returns **1**. For information about collections in variables, see Working with Variables.	**Destination Variable**	Select the variable to receive the return value. If this is a complex variable, you must also select a **Destination Variable Field**.
	Destination Variable Field	If the **Destination Variable** is a complex variable, select the field to receive the return value. If the **Destination Variable** is a simple variable, this list is not populated.
	Source Variable	Select the variable that contains the collection to be counted.
Sort - Takes the contents of a variable or variable field and sorts the collection in ascending or descending order. The variable or variable field must contain a collection for this operation to be meaningful. For information about collections in variables, see Working with Variables.	**Destination Variable**	Select the variable whose contents you want to sort. If the variable is a complex variable, you must select a **Destination Variable Field** as well.
	Destination Variable Field	If the **Destination Variable** is a complex variable, select the field whose contents you want to sort.
	Sort Order	Select one of the following: • **ascending** - Arranges the designated items in ascending order. For example, if the operation is sorting a list of names, the names are arranged in alphabetic order, from A to Z. • **descending** - Arranges the designated items in descending order. For example, if the operation is sorting a list of numbers, the numbers are arranged in order from greatest to least.
Decrement - Subtracts one (1) from the value of the destination variable or variable field. The destination variable must be of the type integer.	**Destination Variable**	Select the variable whose value is to be decremented. If this variable is a complex variable, you must also select a **Destination Variable Field**.
	Destination Variable Field	If the **Destination Variable** is a complex variable, select the field whose value is to be decremented.

Operation Item Property Types (continued)

Type	Associated properties	Comments/Remarks
Increment - Adds one (1) to the value of the destination variable or variable field. The destination variable must be of the type integer.	**Destination Variable**	Select the variable whose value is to be incremented. If this variable is a complex variable, you must also select a **Destination Variable Field**.
	Destination Variable Field	If the **Destination Variable** is a complex variable, select the field whose value is to be incremented.
Next in Collection - Returns the value of the next item in the collection. For information about collections in variables, see Working with Variables.	**Destination Variable**	Select the variable that contains the collection. At run time, when the application reaches the end of the collection and there are no more items to return, the system would normally throw a run-time exception. To prevent this, place a **Condition** item in the node subflow *before* this item, and: • Set the **Operator** property to **HasNoMore**. • For the **Left variable**, select the variable that contains the collection. • For the **Next Form** field, select the next node to go to when there are no more items left in the collection.
Reset Collection - Returns the item marker for the collection to the beginning of the collection. For information about collections in variables, see Working with Variables.	**Destination Variable**	Select the variable whose collection you want to reset to the beginning.

Operation Item Property Types (continued)

Type	Associated properties	Comments/Remarks
Parse - Uses the separator character to break a string into smaller strings. Returns the resulting smaller strings as a collection in the destination variable. For information about collections in variables, see Working with Variables.	**Destination Variable**	Select the variable to which you want to assign the collection that results from the parsing action.
	Source Variable	Select the variable that contains the string you want to parse. If this variable is a complex variable, you must also select a **Source Variable Field**.
	Source Variable Field	If the **Source Variable** is a complex variable, select the variable that contains the string you want to parse.
	Separator Character	Enter the character you want to use as the trigger to break the string into smaller strings. For example, suppose your original string was “Parse this variable,” and you used a space as the separator. This operation returns the three words “Parse”, “this”, and “variable”. It then returns the results as a collection with the three words as items in the collection.
Trim Whitespace - Removes the whitespace from the beginning and end of strings. For example: “ Hello World “ would become “Hello World” after the trim whitespace operation.	**Destination Variable (and Field)**	Select the variable that will contain the results from the trimming action. If the destination is a complex variable, a field must be selected.
	Source Variable (and Field)	Select the variable (and field if Source Variable is a complex variable) that contains the string before the trim operation.
To Upper Case - Converts string values to all upper case.	**Destination Variable**	The variable that receives the upper case converted string.
	Destination Variable Field	The field, if the **Destination Variable** is a complex variable.
	Source Variable	The variable whose value will be converted to all upper case. The converted result is stored in the **Destination Variable**.
	Source Variable Field	The field, if the **Source Variable** is a complex variable.

Operation Item Property Types (continued)

Type	Associated properties	Comments/Remarks
To Lower Case - Converts string values to all lower case.	**Destination Variable**	The variable that receives the lower case converted string.
	Destination Variable Field	The field, if the **Destination Variable** is a complex variable.
	Source Variable	The variable whose value will be converted to all lower case. The converted result is stored in the **Destination Variable**.
	Source Variable Field	The field, if the **Source Variable** is a complex variable.
Index Of - Returns the index of a substring from a larger string.	**Destination Variable (and Field)**	The variable (or field if variable is complex) that stores the index of the “String”. If the “String” value does not exist in the Source Variable (or field if complex), then this value will be -1. **NOTE:** index is 0-based—that is the first character is index 0.
	Source Variable (and Field)	The source variable—the value of this is searched to find the start index of “String” if it exists.
	String	The string to search the source variable for. For example, Index of “Fran” in “San Francisco” would return 4. Index of “Den” in “San Francisco” would return -1.
	Start From Constant **Start From Variable (and Field)**	This is used to skip over preceding characters in the string before searching for the first occurrence of “String”. You can either use a constant value, or supply a variable that contains a valid index to start from. This will throw a runtime exception if the “start from” value is invalid. For example: Index of “a” in “Hawaii” with start from “2” would return 3. Index of “a” in “Hawaii” with start from “10” would throw an exception because the start from index exceeds the length of the string.

Operation Item Property Types (continued)

Type	Associated properties	Comments/Remarks
Set Null - Sets the value of the variable to null. **NOTE:** Session variables cannot be null, so by default, they are initialized as empty strings (“”). In order to pass null values to database operations or web services, “Set Null” will assign a special value (“__DD_NULL”) to the session variable which is interpreted as null by the runtime for web services and databases. Likewise, if a database or web service operation returns a null value, then the runtime will set the value of the session variable to “__DD_NULL” to prevent Null Pointer Exceptions.	**Destination Variable**	Select the variable to be set to Null. If this is a complex variable, a **Destination Variable Field** must also be selected.
	Destination Variable Field	If the **Destination Variable** is a complex variable, select the field to be set to null. If the **Destination Variable** is a simple variable, this list is not populated.

Operation Item Property Types (continued)

Type	Associated properties	Comments/Remarks
Sub String - Returns a part of a string in the source variable and assigns it to the destination variable. The source variable must be of type string. **Note:** The index for a Sub String operation is zero-based. This means that you count the first character as zero (0) and go on from there. For example if you wanted to take the substring "truck" from the word "firetruck," you must use a **Start Index** of **4** (the fifth character) and **Number of Characters** of **5**. Note that, in this case, for **Number of Characters**, you can also enter **-1** (see **Number of Characters** description).	**Destination Variable**	Select the variable to which you want to assign the substring. If this variable is a complex variable, you must also select a **Destination Variable Field**.
	Destination Variable Field	If the **Destination Variable** is a complex variable, select the field to which you want to assign the substring.
	Source Variable	Select the variable from which you want to select and return the substring. If this variable is a complex variable, you must also select a **Source Variable Field**.
	Source Variable Field	If the **Source Variable** is a complex variable, select the field from which you want to select and return the substring.
	Start From **Start From Variable (and Field)**	Enter in this field the point in the string at which the substring starts. For example, if you want the substring to start at the seventh character in the string, enter **6**. You can use a variable for the Start From rather than a hard coded value. If Start From variable is complex, then you need to provide a field.
	Number of Characters **Num Chars Variable (and Field)**	Enter in this field the maximum number of characters to include in the substring that is returned. If the string has fewer than this number, it returns a shorter substring. If you enter **-1** in this field, the operation returns all characters starting with and following the index. So, for example, if you wanted to remove the first three digits from an account number that might have a variable number of digits, you would set the **Start Index** to **3** and set the **Number of Characters** to **-1**. The default for this field is **-1**. You can use a variable for the number of characters to extract rather than a constant. If the Num Chars variable is complex, then you must select a field.

Operation Item Property Types (continued)

Type	Associated properties	Comments/Remarks
Remove Character - Removes all occurrences of a designated character from the source variable and writes the new value of the variable to the destination.	**Destination Variable**	Select the variable to which you want to write the new value. If this variable is a complex variable, you must also select a **Destination Variable Field**.
	Destination Variable Field	If the **Destination Variable** is a complex variable, select the field to which you want to write the new value.
	Source Variable	Select the variable from which you want to remove all occurrences of the designated character. If this variable is a complex variable, you must also select a **Source Variable Field**.
	Source Variable Field	If the **Source Variable** is a complex variable, select the field from which you want to remove all occurrences of the designated character.
	Character	Enter in this field the character you want to remove from the source variable or variable field.

Operation Item Property Types (continued)

Type	Associated properties	Comments/Remarks
Each of the following types performs an arithmetic operation on two number values and returns the result to the destination variable. **Add** - Adds the value of the left operand and the right operand. **Subtract** - Subtracts the value of the right operand from the left operand. **Multiply** - Multiplies the value of the left operand and the right operand. **Divide** - Divides the value of the left operand by the right operand. **Modulus** - Divides the value of the left operand by the right operand and returns the value of the remainder, if any.	**Destination Variable**	Select the variable to which you want to assign the return value of the operation. If this variable is a complex variable, you must also select a **Destination Variable Field**.
	Destination Variable Field	If the **Destination Variable** is a complex variable, select from the drop-down the field to which you want to assign the return value of the operation.
	Left Operand Variable	Select the variable to use as the left operand. If this variable is a complex variable, you must also select a **Left Operand Variable Field**. If you use this property, you cannot also use the **Left Operand Constant** property.
	Left Operand Variable Field	If the **Left Operand Variable** is a complex variable, select the field to use as the left operand.
	Left Operand Constant	Enter in this field the integer value to use as the left operand. If you use this property, you cannot also use the **Left Operand Variable** property.
	Right Operand Variable	Select the variable to use as the right operand. If this variable is a complex variable, you must also select a **Right Operand Variable Field**. If you use this property, you cannot also use the **Right Operand Constant** property.
	Right Operand Variable Field	If the **Right Operand Variable** is a complex variable, select the field to use as the right operand.
	Right Operand Constant	Enter in this field the integer value to use as the right operand. If you use this property, you cannot also use the **Right Operand Variable** property.

Operation Item Property Types (continued)

Type	Associated properties	Comments/Remarks
Append To Collection - Appends the value of a source variable to the end of a collection in the destination variable. For information about collections in variables, see Working with Variables.	**Destination Variable**	Select the variable containing the collection to which you want to append the new entry.
	Source Variable	Select the variable containing the value you want to append to the collection.
Insert Into Collection - Inserts the value of a source variable into a collection in the destination variable at the current position of the collection pointer. For information about collections in variables, see Working with Variables.	**Destination Variable**	Select the variable containing the collection into which you want to insert the new entry.
	Source Variable	Select the variable containing the value you want to insert into the collection.
Delete From Collection - In a collection, deletes the entry that is at the current position of the collection pointer. For information about collections in variables, see Working with Variables.	**Destination Variable**	Select the variable containing the collection from which you want to delete the entry.

Output Parameter 

Type

Application item

Available from

Return Node only

Purpose

When using a modular approach to application design, it is often necessary to pass variable or constant values from one module to another. The **Output Parameter** item makes it possible for a speech project used as a module node to pass such data back to the parent speech project. For more information about creating and using application modules, see Module Nodes.

Behavior

Use the **Output Parameter** item when you want to pass a variable value from a module node to the parent speech project. To pass values from the module back to the parent application, you must place into the Return node one **Output Parameter** item for each value that you want to pass back.

When you are satisfied that your module works the way you want it to work, generate and save it. You must then deploy it as a Dialog Designer module. For more information about this and the procedure to deploy it, see Deploying Projects as Dialog Designer Modules.

Later, when you use this speech project as a module in another project, Dialog Designer automatically populates the module node with one **Module Output** item for each **Output Parameter** item in the module. See Module Output.

Properties

- **Name** - Enter the name you want to identify the value being passed out of the module.

  Dialog Designer assigns a default name automatically when you place the **Output Parameter** item in the node, and you can leave it as is or rename it.

  Note:
  This name must follow conventions for naming Java components as described in Naming Java Components.

- **Description** - (Optional) Enter in this field a brief description of what the output parameter value is used for or represents. Although this is optional, Avaya recommends that you write a good description. The reason is that the description can be helpful as a reminder of what the output parameter is for, later on when you or others use the module in another speech project.

- **Variable** - Select the variable whose value you want to pass back to the parent application. If the variable you select is a complex variable, you must also select a **Variable Field**.
- **Variable Field** - If the variable you selected in the **Variable** field is a complex variable, you must select from this drop-down list the field whose value you want to pass back to the parent application. If the variable you selected in the **Variable** field is a simple variable, this field is blank.

Phrase

Type

Prompt Segment item

Available from

Any level tab in the Prompt File Editor. For more information about the Prompt File Editor, see Using the Prompt File Editor.

Purpose

The Phrase item plays the selected audio file in a prompt. This item makes it possible to use prerecorded audio files in prompts.

Behavior

When the system plays a prompt containing a Phrase item, it calls and plays the designated audio phrase file in its turn.

Before the Phrase item is functional, you must create the audio phrase file to use in it. For more information about creating audio phrase files, see Working with Phrases.

Properties

- **Phraseset** – Name of the phraseset that contains the phrase that you want to play.
- **Phrase in phraseset** – Name of the phrase within the selected phraseset.
- **Standalone Phrase** – Name of phrase that you want to play (if standalone).

Phrase Variable

Type

Prompt Segment item

Available from

Any level tab in the Prompt File Editor. For more information about the Prompt File Editor, see Using the Prompt File Editor.

Purpose

The **Phrase** variable Item can play an audio phrase referred by a phraseset or a standalone phrase or an audio URL.

Behavior

A **Phrase** variable can point to a phrase in phraseset, a standalone phrase or an external audio file. For more information about creating audio phrase files, see About Phrasesets.

Properties

- **Type** - When the Type is Phraseset:Phrase ID, you need to supply in the Variable Field a Variable name, whose format should be Phraseset:Phrase ID. When the Type is Phrase ID, you need to supply in the Variable Field a variable name, which refers to the standalone phrase name. When the Type is Audio URL, you need to supply in the Variable Field a variable name which contains the external audio file URL.
- **Variable** - A variable name which points to a phrase in phraseset, a standalone phrase or an external URL.
- **Variable Field** - A field contained by a Variable. If you use simple Variable, leave this field empty.
- **Backup Text** - The alternative text played if DD cannot find the audio file in the phrase or phraseset.
- **Backup Variable** - The Variable which contains the alternative text played if DD cannot find the audio file in the phrase or phraseset.
- **Backup Field** - A field contained by a Backup Variable. If you use simple Variable, leave this field empty.

Prompt 

Type

Form item

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Announce Node
- Prompt and Collect Node

- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Disconnect Node
- Menu Node
- Form Node
- Return Node

Purpose

The **Prompt** item makes it possible to play a selected prompt. For example, in a Prompt and Collect node, use the **Prompt** item to prompt the caller for a response. In a Record node, use the **Prompt** item to instruct the caller how to make the recording and what the post-recording options are.

Behavior

When placed in a node, the **Prompt** item is generally played in the order it appears with other Prompt items at the same level within the node. Also, higher level **Prompt** items are played before lower level **Prompt** items.

For example, suppose you have five **Prompt** items within a single Form node. Two of those **Prompt** items are assigned to an **Input** item. The subflow of the node is as follows:

- Prompt 1
- Prompt 2
- Input
  - Prompt 3
  - Prompt 4
- Prompt 5

In this example, the system would play the prompts in the following order: 1, 2, 5, 3, 4. The reason for this play order is that the higher level prompts (1, 2, and 5) are played in order before the lower level prompts (3 and 4).

If a **Prompt** item is a sub-item to another item, however, it plays before the item is executed. For example, if you place a **Prompt** item as a sub-item to a **Blind Transfer** item, the system plays the prompt before transferring the call.

In the **AppRoot** node, the **Prompt** item can only be used as a sub-item to one of the event handlers.

Note:
Before you can select and use a prompt in a **Prompt** item, you must first create and save the prompt.

Properties

- **Name** - Select the prompt you want to play.

Property 

Type

Form item

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Menu Node
- Form Node

Purpose

The **Property** item is used to set one or more values that affect system behavior. These values include properties like timeouts and recognition processes.

Behavior

When placed in the AppRoot node, the **Property** item sets application-wide defaults for the designated properties. When placed in a node, the **Property** item sets the designated properties at the node level. When attached to a node item, the **Property** item sets the designated properties at the item level.

When a given property is set using the **Property** item, that setting affects anything below it in the application. For example, if you set the **Timeout Complete** property in a node, that setting affects everything that comes after it in the node. It does not, however, affect other nodes. At a given level, the property value remains in effect until it is reset at a later point. For example, if you have a node with two property value settings for a particular property within the same node, the first stays in effect until the application reaches the second property setting.

A property setting at a lower level overrides a property setting for the same property at a higher level. For example, if you have the DTMF **Terminating Character** property set in both the AppRoot node and another DTMF **Terminating Character** property set in a node, the property at the node level overrides the one set at the AppRoot node level.

You can set as many or as few property settings as you want in a single **Property** item.

Note:
To set properties that are not defined within the Property item, use the External Property item.

Properties

The properties for the **Property** item are separated into three general groups according to their functions:

- **Speech** - These property settings determine various aspects of how the ASR engine responds to spoken inputs.

Speech (ASR) Property Setting	Description
Confidence Level	Enter the minimum required level of confidence for an ASR engine to return a recognition of a speech utterance. The system rejects any utterance that returns a confidence level lower than this value and throws a No Match event. The range of valid values for this property is 0.0 through 1.0. A value of 0.0 means that minimum confidence is required for a recognition. A value of 1.0 means that a maximum confidence is required for a recognition. Default = **0.5**
Sensitivity	Enter the sensitivity level for speech recognition to start. This setting determines how loud an utterance must be for the ASR engine to start speech recognition. The range of valid values for this property is 0.0 through 1.0. The higher the number you enter in this field, the more sensitive the ASR engine is, and the less amount of input volume it takes to start speech recognition. Default = **0.5**
Speed vs. Accuracy	Enter the number to determine the balance between speed of recognition and accuracy of recognition. The range of valid values for this property is 0.0 through 1.0. A lower value tips the balance in favor of speed of recognition. A higher value tips the balance in favor of accuracy of recognition. Default = **0.5**

Speech (ASR) Property Setting	Description
Timeout Complete	Enter the number of seconds of silence the system must wait before finalizing an utterance. Note that, if no utterance was ever recognized with a match, the system uses this timeout period to determine when to throw a No Match event. If an utterance was successfully recognized, this is the period of silence the system waits before returning the result. Valid values for this property are positive real numbers. Numbers can be of the format "*n*", "*n*.", ".*n*" or "*n.n*", where *n* represents any sequence of one or more digits. Take care when setting this property. If you set the number too high, it can slow down system response time, even for successful recognitions. If you set the number too low, it can lead to an utterance being cut off prematurely. Reasonable values for this property are usually in the range 5 to 15 seconds. This property has no default value. The default value for this is set on the ASR server. If you set a value in this field, this value overrides the default on the ASR server. See also the next property, **Timeout Incomplete**.

Speech (ASR) Property Setting	Description
Timeout Incomplete	Enter the number of seconds of silence the system must wait before finalizing an incomplete recognition of an utterance. An incomplete recognition is defined as an incomplete match of all active grammars. In this case, once the timeout is reached, the system rejects the utterance and throws a No Match event. An incomplete timeout can also occur when there is a complete match of an active grammar but it is possible to speak further and still match the grammar. By contrast, a complete timeout (see previous property, **Timeout Complete**) takes place when there is a complete match of an active grammar and no further input can be recognized. Valid values for this property are positive real numbers. Numbers can be of the format "*n*", "*n*.", ".*n*" or "*n.n*", where *n* represents any sequence of one or more digits. Take care when setting this property. If you set the number too high, it can slow down system response time, even for successful recognitions. If you set the number too low, it can lead to an utterance being cut off prematurely. This property is usually set lower than the **Timeout Complete**, but high enough to allow callers to pause mid-utterance, for instance, for a breath. Reasonable values for this property are usually in the range 2 to 5 seconds. This property has no default value. The default value for this is set on the ASR server. If you set a value in this field, this value overrides the default on the ASR server. See also the previous property, **Timeout Complete**.
Maximum Speech	Enter the number of seconds you want to allow the caller to continue speaking before the system ends the recognition attempt and throws a **maxspeechtimeout** event. Valid values for this property are positive real numbers. Numbers can be of the format "*n*", "*n*.", ".*n*" or "*n.n*", where *n* represents any sequence of one or more digits. This property has no default value. The default value for this is set on the ASR server. If you set a value in this field, this value overrides the default on the ASR server.

- **General** - These property settings determine various aspects of how the system responds to DTMF (touchtone) key press inputs.

General Property Setting	Description
Interdigit timeout	Enter the number of seconds the system must wait for an additional DTMF input when such an input is expected. This property becomes active when a DTMF expects and requires a certain number of DTMF input digits. For example, when collecting a telephone number from a caller, the DTMF grammar might be set up to require a ten-digit number. If the caller enters eight digits and then stops, this property is triggered at the end of the timeout setting, and the system throws a No Match event. Valid values for this property are positive integers. This property has no default value. The default value for this is set on the server that handles DTMF responses. If you set a value in this field, this value overrides the default on that server.
Privacy	When enabled, this property will omit certain pieces of data from the logs. For example, a user's credit card number might be private information that should be suppressed from the logs. When enabled, it enables the privacy option on the Voice Portal (only; this property is not understood by IR). For more information, see the Voice Portal documentation.

General Property Setting	Description
Terminating timeout	Number of seconds the system must wait for an additional DTMF input when such an input is possible. This property becomes active when a DTMF can accept a variable number of DTMF input digits. For example, when collecting an account number from a caller, the account might be either seven or eight digits long. If the caller enters seven digits and then stops, this property is triggered at the end of the timeout setting, and the system returns a successful recognition result. Valid values for this property are positive real numbers. Numbers can be of the format "*n*", "*n*.", ".*n*" or "*n.n*", where *n* represents any sequence of one or more digits. This property has no default value. The default value for this is set on the server that handles DTMF responses. If you set a value in this field, this value overrides the default on that server.
Terminating character	The DTMF key press that the system must use to end a DTMF input. When this key is pressed, the system ends the DTMF input recognition and returns the appropriate result. Valid values for this property are the numbers **0** through **9**, the star (*****) key, and the pound (#) key. The default is #

- **Fetch** - These property settings pertain to the fetching of new documents and resources.

Fetch Property Setting	Description
fetchTimeout	The timeout for fetches. The value is a Time Designation. The default value is platform-dependent.
grammarmaxage	Tells the platform the maximum acceptable age, in seconds, of cached grammars. The default is platform-specific.
grammarmaxstale	Tells the platform the maximum acceptable staleness, in seconds, of expired cached grammars. The default is platform-specific.

- **Transitional Audio** - These property settings determine what prompt is to be used as a transitional audio prompt and how it is to behave. For more information about transitional audio prompts, what they are used for, and how they behave, see About Transitional Audio Prompts.

Transitional Audio Property Setting	Description
Audio Prompt	Select the prompt to use as the transitional audio prompt. Note that a prompt must be defined as a transitional audio prompt when it is created. Only transitional audio prompts can be selected for this property.
Delay seconds	Enter the number of seconds the system must wait before starting to play the transitional audio prompt. The idea is that, if the system response time is short enough, it might be better to have a short period of silence, rather than starting to play an audio file that is immediately cut off. Valid values for this property are positive real numbers. Numbers can be of the format “*n*”, “*n*.”, “.*n*” or “*n.n*”, where *n* represents any sequence of one or more digits. This property has no default value. The default value for this is set on the IVR system. If you set a value in this field, this value overrides the default on the IVR system.
Minimum play seconds	Enter the minimum number of seconds of the prompt that you want the system to play before proceeding. The idea is that, once the caller does begin to hear the prompt, it should not be stopped too quickly. This can be useful, for instance, when there is a short message you want callers to hear before the system proceeds. Valid values for this property are positive real numbers. Numbers can be of the format “*n*”, “*n*.”, “.*n*” or “*n.n*”, where *n* represents any sequence of one or more digits. This property has no default value. The default value for this is set on the IVR system. If you set a value in this field, this value overrides the default on the IVR system.

Prosody 

Type

SSML item

Available from

Any level tab in the Prompt File Editor. For more information about the Prompt File Editor, see Using the Prompt File Editor.

Purpose

The **Prosody** item is a Synthesized Speech Markup Language (SSML) element that makes it possible to control various aspects of Text-to-Speech (TTS) synthesis. This element means you can get more natural-sounding speech synthesis from the TTS engine.

Behavior

Based on the properties you set, the **Prosody** item alters the rendering of TTS speech synthesis. All properties are optional, but if you use the **Prosody** item, you must use at least one property. The **Prosody** item has six basic properties:

Note:
The specific effects of many of these properties might vary from one TTS engine to another.

- **Pitch** - This property controls the baseline pitch of the speech output. Increasing this property raises the baseline pitch. Decreasing this property lowers the baseline pitch.
- **Range** - This property controls the pitch range, or variability, of the speech output. Increasing this property increases the range of pitches the TTS engine produces. Decreasing this property decreases the pitch range.
- **Rate** - This property controls the speaking rate at which the TTS engine produces the output. Increasing this property speeds up the synthesized speech. Decreasing this property slows down the synthesized speech.
- **Duration** - This property is the desired amount of time it takes the TTS engine to read the contents of the TTS element. The value is stated in seconds (s) or milliseconds (ms). The text to be spoken is either compressed or expanded to fit into the duration you select. For example, if the text would normally take five seconds to speak, you set this property to four seconds, the system increases the rate to make the reading of the text fit into four seconds.

  The **Duration** property takes precedence over the **Rate** property.
- **Volume** - This property controls the volume, or loudness, of the speech output. Increasing this property makes the TTS output louder. Decreasing this property makes the TTS output quieter.

- **Contour** - This property sets the actual pitch contour for the speech output. This contour is accomplished by means of a series of percent/pitch value pairs. For details, see the next section, “Properties.”

  The **Contour** property takes precedence over both the **Pitch** and **Range** properties.

For additional details on the properties and how they behave, see the next section, “Properties.”

Note:
The **Prosody** item and its properties function correctly only with SSML-compliant speech synthesis engines. The Microsoft Speech SDK, which is used by Dialog Designer during application simulation, is *not* an SSML-compliant speech synthesis engine, so any settings you make with this item are ignored. For more information about the SSML standard, see the Speech Synthesis Markup Language (SSML) Version 1.0 W3C Recommendation.

Properties

All properties are optional, but if you use the **Prosody** item, you must use at least one property. Note that all units, such as **Hz** and **st**, are case-sensitive.

- **Pitch** - Select one of the following settings:

Pitch Setting	Description
default	This setting has no effect on the output. This setting uses the default baseline pitch of the TTS server.
x-low **low** **medium** **high** **x-high**	These settings represent a range of pitch options. The specific application of these properties varies according to the TTS server.

Pitch Setting	Description
custom	With this setting, you can define the baseline pitch that you want, as a modification of the TTS server default. When you select this setting, Dialog Designer automatically adds the **Custom Pitch [Hz or st]** property to the Property view.
Custom Pitch [Hz or st]	This setting is available only when the **custom** setting for **Pitch** is selected. With it, you can fine tune the baseline pitch by raising or lowering the pitch relative to the server default: • To raise or lower the pitch based on frequency, enter a number followed by **Hz**. For example, if you set this to **+8000Hz** the system raises the baseline pitch by 8000 cycles per second. If you set this to **-500Hz**, the system lowers the pitch by 500 cycles per second. • To raise or lower the pitch by semitones, enter a positive or negative number followed by **st**. Each whole number represents a semitone on the diatonic scale. Positive numbers raise the pitch. Negative numbers lower the pitch. For example, if you enter **+3st** in this field, the baseline pitch is raised by three semitones. If you enter **-1.5st**, the baseline pitch is lowered by one and a half semitones. The correct format to enter these values is a positive (+) or negative (-) sign, followed by a number, followed by **Hz** or **st**, with no spaces. Numbers can be of the format "*n*", "*n*.", ".*n*" or "*n.n*", where *n* represents any sequence of one or more digits.

- **Range** - Select one of the following settings:

Range Setting	Description
default	This setting has no effect on the output. This setting uses the default pitch range of the TTS server.
x-low **low** **medium** **high** **x-high**	These settings represent a range of pitch range options. The specific application of these properties varies according to the TTS server.

Range Setting	Description
custom	With this setting, you can define the pitch range that you want, as a modification of the TTS server default. When you select this setting, Dialog Designer automatically adds the **Custom Range [Hz or st]** property to the Property view.
Custom Range [Hz or st]	This setting is available only when the **custom** setting for **Range** is selected. With it, you can fine tune the pitch range by increasing or decreasing the range relative to the server default: • To increase or decrease the pitch range based on frequency, enter a number followed by **Hz**. For example, if you set this to **+8000Hz** the system increases the pitch range by 8000 cycles per second. If you set this to **-500Hz**, the system lowers the pitch range by 500 cycles per second. • To increase or decrease the pitch by semitones, enter a positive or negative number followed by **st**. Each whole number represents a semitone on the diatonic scale. Positive numbers increase the pitch range. Negative numbers decrease the pitch range. For example, if you enter **+3st** in this field, the pitch range is increased by three semitones. If you enter **-1.5st**, the baseline pitch is decreased by one and a half semitones. The correct format to enter these values is a positive (+) or negative (-) sign, followed by a number, followed by **Hz** or **st**, with no spaces. Numbers can be of the format "*n*", "*n*.", ".*n*" or "*n.n*", where *n* represents any sequence of one or more digits.

- **Rate** - Select one of the following settings:

Rate Setting	Description
default	This setting has no effect on the output. This setting uses the default speaking rate of the TTS server.
x-slow **slow** **medium** **fast** **x-fast**	These settings represent a range of speaking rate options. The specific application of these properties varies according to the TTS server.

Rate Setting	Description
custom	With this setting, you can define the speaking rate that you want, as a modification of the TTS server default. When you select this setting, Dialog Designer automatically adds the **Custom Rate [positive float]** property to the Property view.
Custom Rate [positive float]	This setting is available only when the **custom** setting for **Rate** is selected. With it, you can fine tune the speaking rate by increasing or decreasing the rate relative to the server default. The number you enter in this field acts as a multiplier on the default rate. For example, a setting of **1** or **100%** in this field means there is no change to the default rate. A setting of **2** or **200%** in this field makes the speaking rate twice as fast as the default rate. A setting of **0.5** or **50%** in this field makes the speaking rate half as fast as the default rate. The correct format to enter these values is "*n*", "*n*.", ".*n*" or "*n.n*", where *n* represents any sequence of one or more digits.

- **Duration** - Select one of the following settings:

Duration Setting	Description
250ms **500ms** **750ms** **1s** **2s** **3s** **4s** **5s**	These settings represent a range of duration options in milliseconds (ms) or seconds (s). This setting overrides any **Rate** setting you might have.
custom	With this setting, you can define the exact duration that you want. When you select this setting, Dialog Designer automatically adds the **Custom Duration [s or ms]** property to the Property view.
Custom Duration [s or ms]	This setting is available only when the **custom** setting for **Duration** is selected. With it, you can set the exact duration for the text to be spoken. The correct format to enter these values is "*n*", "*n*.", ".*n*" or "*n.n*", where *n* represents any sequence of one or more digits. The number must be followed, with no space in between, by **s** for seconds or **ms** for milliseconds.

- **Volume** - Select one of the following settings:

Note:
The **Volume** property uses a range of 0.0 (silent) to 100.0 (full volume).

Volume Setting	Description
default	This setting is the same as setting the volume to 100.0, or full volume.
silent	This setting is the same as setting the volume to 0.0.
x-soft **soft** **medium** **loud** **x-loud**	These settings represent a range of volume options. The specific application of these properties varies according to the TTS server.
custom	With this setting, you can define the exact volume that you want. When you select this setting, Dialog Designer automatically adds the **Custom Volume [float]** property to the Property view.
Custom Volume [float]	This setting is available only when the **custom** setting for **Volume** is selected. With it, you can set the exact volume you want for the text to be spoken. The correct format to enter these values is a positive (+) or negative (-) sign, followed by a number. Numbers can be of the format “*n*”, “*n.*”, “*.n*” or “*n.n*”, where *n* represents any sequence of one or more digits.

- **Contour** - Enter in this field the pitch contour settings you want to use for the text to be spoken.

  A pitch contour is defined by a set of *time-position/target-pitch* pairs. These pairs must in the form: **`(time-position,target-pitch)`**. The *time-position* value is a percentage of the total time period for the text to be spoken. The *target-pitch* value is the relative pitch for the text to be spoken at that point in time. The algorithm for interpolating the pitch from one target to the next is specific to each TTS engine.

  For example, suppose you want the TTS server to speak the phrase, “Good morning.” In normal conversation, the relative pitch of the syllable “morn-” is higher than the word “Good.” The pitch of the final syllable, “-ing,” then, drops below the initial word “Good.” So, to create a pitch contour for this phrase, you might create the following (*time-position*, *target-pitch*) pairs:

  ```
  (0%,+20Hz)(40%,+30%)(75%,-10Hz)
  ```

  In this example, the initial pitch for the phrase, and the word “Good,” is set at 20 cycles per second above the baseline pitch. At a time position 40% of the way into the phrase, the pitch rises to 30% above where the phrase started. This takes place at about the point where the server speaks the syllable “morn-”. The final syllable, “-ing,” starts about 75% of the way into the phrase, in terms of time. This final syllable drops to a point, then, 10 cycles per second below the baseline pitch.

  For more information about the Contour property of the Prosody item, see the Speech Synthesis Markup Language (SSML) Version 1.0 W3C Recommendation.

Record 

Type

Form Item

Available from

- Announce Node
- Record Node
- Form Node

Note:

The **Record** item cannot be used in the same node as the **Blind Transfer** item, the **Bridged Transfer** item, or the **Input** item. So, although the **Record** item appears on palettes for other nodes, such as the Blind Transfer node, the Bridged Transfer node, and the Prompt and Collect node, it is not actually available for use in those nodes without destroying their intended functionality. It is intended primarily for use in the Record node.

Purpose

There are times when you want to collect information from callers that is not easily collected using DTMF or ASR input. For example, you might want to get the address of the caller, to enter into a database later on. Or you might want to allow callers to leave voice messages for the parties they are trying to reach. The **Record** item is the call flow element that is responsible for collecting recorded input and saving it to a file for retrieval later.

Behavior

The **Record** item:

- Optionally, plays a prompt to the caller, instructing the caller what to do and how to signal that the recording is finished.
- Optionally, plays a beep tone to signal the start of the recording.
- Records the message from the caller, starting after the tone, if the beep is turned on. If the beep is not turned on, the **Record** item starts immediately after the prompt, if there is one. If there is no prompt, the system starts recording as soon as the **Record** item starts to execute.

  Note:
  If the caller does not start recording within the time as set in the **Max Silence** property, the system throws a **No Input** event. For more information about **Record** item properties, see the following section, “Properties.”

- Ends the recording when one of the following conditions is met:
  - A predetermined period of silence.
  - The caller signals with a DTMF key press that the caller is done recording.
  - The recording reaches the maximum allowed period of time for a recording.
  - The caller hangs up.
  - The system throws an event from which the application cannot continue.
  - The caller utters something or enters a DTMF key press that triggers an active grammar.
- Stores the recording as an audio file.
- Returns a series of variable values to the application, based on data taken from the recording session.

When you place this item in a node subflow, Dialog Designer automatically creates a complex variable with the same name. This variable contains the following fields, which are populated upon completion of the node at run time:

- **confidence** - This field returns a value only if the caller uses an active grammar “hotword” to terminate the recording. When that happens, this field returns a value that indicates the ASR server confidence level that the hotword was uttered and recognized. Note that not all IVR systems support simultaneous speech recognition and recording, in which case this field is never populated.

- **duration** - This variable field is the length of time that the recording lasted, in milliseconds. For example, if the recording lasted for just over 24 seconds, this might have a value of **24389**.
- **maxtime** - If the caller exceeded the maximum allowable time for the recording, the system automatically ends the recording. If this happens, the system returns a value of **true** to this variable field. Otherwise, this variable field has the default value of **false**.
- **size** - This field is the size of the recorded audio file, in bytes.
- **termchar** - The value of this field is the DTMF character, if any, that was used to end the recording. If the caller did not end the recording with a DTMF key press, this variable field is undefined.
- **utterance** - This field returns a value only if the caller uses an active grammar “hotword” to terminate the recording. When that happens, this field returns a value that indicates what the caller said that the ASR server recognized as being the hotword to terminate the recording. Note that not all IVR systems support simultaneous speech recognition and recording, in which case this field is never populated.
- **value** - This variable field contains the URL to the recorded audio file.

To play the audio file of the recording back later in the application:

1. Create a prompt in which to play the audio file.
2. Place a text variable in the prompt.

   For more information about text variables, see Text Variable.
3. In the **Variable** field for the text variable, select the **Record** item variable.

   Note:
   The **Record** item variable has the same name as the **Record** item.
4. In the **Variable Field** field for the text variable, select **value**.
5. In the **Format** field for the text variable, select **url**.

The **value** field of the **Record** item variable contains the URL of the audio file that was recorded. When the Avaya Application Simulator comes to this variable field which uses the URL format, the system retrieves the audio file from the designated URL and plays it back.

You can also use other functions to assign the recorded audio file to a database, from which you can retrieve it later. This might be done, for instance, to have an administrator later retrieve an address and convert it to a text-based format for later use.

Properties

- **Name** - Enter the name you want to assign to the **Record** item and its associated variable.

  If you do not enter a name, Dialog Designer automatically assigns both the item and the variable a default name.

- **Play Beep** - If you want the system to play a short beep before starting the recording, set this field to **true**, which is the default. If you do not want the system to play a beep, set this field to **false**.
- **Max Length** - Enter the number of seconds that you want to allow for the maximum duration of a recording. For example, if you want to allow callers a maximum of two minutes of recording time, set this to **120**.

  The maximum allowable value for this field depends on the limitations and settings on the IVR system. The default is **60**.
- **Max Silence** - Enter the number of seconds to allow silence at the end of the recording before the system ends the recording. When the system detects a period of silence, the system ends the recording after it reaches this number of seconds of silence.

  If no recording was ever started, the system throws a No Input event when it reaches this number of seconds of silence.

  The default for this field is **3**.
- **Modal** - To deactivate any non-local grammars while a recording is in progress, set this to **true**, the default. To allow any non-local grammars to remain active while the recording is in progress, set this field to **false**.

  Often, any global grammars you have active are used to throw events from anywhere in the application. With this setting, you can override those global grammars while a recording is in progress. This means that something the caller says does not accidentally throw an event and prematurely end the recording.
- **DTMF Terminate** - To allow callers to end the recording by pressing a touch tone key on the key pad, set this to **true**. If this property is set to **true**, the system recognizes any DTMF key press that is not part of a local DTMF grammar as a signal to end the recording.

  To cause the system to ignore DTMF key presses that are not specified in a local DTMF grammar, set this to **false**.

  Note:
  If you set this property to **true**, the system treats it, essentially, as a local grammar. This setting overrides any other local DTMF grammars that might be active.
- **Bind Name to Node** – (true/false). When true, the name of the variable item is bound to the name of the node. When the parent node is renamed, the name of the variable item is renamed as well; keeping the name of the variable in sync with the node name.

- **Audio Type** - Select one of the following formats:

Audio format	Description
audio/x-wav	WAV format: This format is a WAV (RIFF header) 8kHz 8-bit mono mu-law or A-law [PCM] single channel format.
audio/x-alaw-basic	G.711-compliant format: This format is a raw (headerless) 8kHz 8-bit mono A-law [PCM] single channel format.
audio/basic	G.711-compliant format: This format is a raw (headerless) 8kHz 8-bit mono mu-law [PCM] single channel format.

Report 

Type

Tracking item

Available from

- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Disconnect Node
- Menu Node
- Form Node
- Data Node
- Tracking Node

Purpose

The primary purpose of the **Report** item is to track and submit application data that you can use to analyze and evaluate application performance in the run-time environment. You can use the **Report** item to:

- Define the type of data the item is reporting.
- Declare the severity of alarm that is associated with each item.
- Give the reason the item is being reported.

- Record the value of variables or variable fields at that point in the application.

Behavior

During simulation or run time, the **Report** item takes the value of the various fields and properties and writes these values to a log file. The application locates this log file in the ***ProjectName*/data/log** directory, where ***ProjectName*** represents the name of the speech application project.

To view the log file contents after application simulation in Dialog Designer, locate the appropriate log file in this directory and double-click it.

Note:
If you are using an Avaya Voice Portal system, you can view an automatically generated Application Report of this log data. See “Application Report” in your Voice Portal documentation.

Tip:
The name of the log file for the current day is always **report.log**. Each time you run a simulation, Dialog Designer appends the log data to the end of this file. After the end of each day on which data has been written to the log file, Dialog Designer renames the log file to: **report.log.*yyyy-mm-dd***, where ***yyyy-mm-dd*** represents the date on which the log file was compiled.

Report log data is written using ***log4j*** logging technology. Because this is new technology and available information constantly changes, for more information about log4j technology, Avaya recommends that you perform an Internet search with the search term “log4j”.

Dialog Designer applications are set at deployment whether they are to operate in an Avaya Interactive Response (IR) or an Avaya Voice Portal environment.

- If in an IR environment, the **Report** item data is written to a log file on the application server. This makes it easier to use standard reporting or database software to retrieve the information for reports which you can analyze more easily.
- If in a Voice Portal environment, the **Report** item data is written to an SQL database on the Voice Portal Management System (VPMS) server. The Voice Portal Management System administration tool provides an option to generate and view an Application Report that uses this data. See “Application Report” in your Voice Portal documentation library.

To view the log file contents in a run-time environment, perform one of the following actions:

- For IR systems, locate the appropriate log file on the application server, and view it with your text viewer or word processor of choice.

  To locate the file, start with the directory in which you have the Tomcat server software installed. From there, use the following path to the log file:

  ***TomcatHome*/webapps/*applicationName*/data/log**

  where ***TomcatHome*** is the directory in which you installed the Tomcat software and ***applicationName*** is the name of your application.

- For Voice Portal systems, from the main menu, select **Reports** > **Application**. For more information about this report tool, see “Application Report” in your Voice Portal documentation library.

Because the **Report** item is considered primarily a reporting tool for analysis and evaluation of applications, there is no need to be able to enable or disable it. When you use it in an application, therefore, it is always enabled.

Properties

- **Type** - Select the type you want to assign to this Report item.

  Dialog Designer writes the type you select to the log file as part of the report data. This makes it possible to search the log file or database for specific types with which to perform an analysis.

  For example, suppose you wanted to analyze the success rate of a car rental application in getting customers to reserve a rental car over the telephone. You can place a **Report** item at the start of the actual transaction part of the call flow and select **Start** for the type here.

  Then, at the end of the rental transaction, when the customer has verified and approved the reservation, you can place another **Report** item and select **End** for the type.

  Later, after the application has been in service for some time, you can retrieve the report data and compile it into a report that compares the number of transactions that were started versus the number of transactions that were successfully completed.

  Note:
  For this example to work, you must be careful to use the same name in the **Name** field for both the **Start Report** item and the **End Report** item.

  The following table lists and describes the types available.

Type	Use this alarm descriptor to indicate:
Start	Use this type to indicate the beginning of a section of the call flow on which you want to report data. To be used effectively, the **Name** field for this **Report** item must exactly match the **Name** field of a **Report** item of type **End** that comes later in the call flow.
End	Use this type to indicate the end of a section of the call flow on which you want to report data. To be used effectively, the **Name** field for this **Report** item must exactly match the **Name** field of a **Report** item of type **Start** that occurs earlier in the call flow.
Cancel	Use this type to indicate a cancellation of an **Report** item of type **Start** that occurs earlier in the call flow. To be used effectively, the **Name** field for this **Report** item must exactly match the **Name** field of a **Report** item of type **Start** that precedes it in the call flow.

Type	Use this alarm descriptor to indicate:
Cancel All	Use this type to indicate a cancellation of all **Report** items of type **Start** that occur earlier in the call flow.
In Progress	Use this type to report data on a type already in progress. In other words, use this type somewhere between a **Start** type and an **End** type to collect data in the interim.

- **Level** - Select the level of alarm you want to generate with this **Report** item.

  Dialog Designer writes the alarm level you select to the log file as part of the report data. This makes it possible to search the log file for specific problems based on the alarm level of the report entry. It also makes it possible to use other software to generate alarms on the system by reading and responding to the contents of the **`report.log`** file.

  Tip:

  In Avaya IR systems, to do this, you must write or acquire an application to monitor the log file and then generate a system alarm when certain key words are encountered. In Avaya Voice Portal systems, to do this you must write or acquire a similar application to monitor the database to which the data is being written.

  The following table lists the alarm levels for **Report** items, shows how they appear in the **report.log** file, and provides a brief description of what each level means.

Alarm Level	Description
Fatal	A situation in which the application would crash.
Error	A situation that would cause the application to generate an error.
Warning	A situation that would cause the application to generate a warning.
Info	A situation that would cause the application to generate an informational message.

  Tip:

  On Voice Portal systems, the Application Report displays this setting for entries in the **Level** column.

- **Activity Name** - Enter in this field the name you want to use to identify the **Report** item entry in the **report.log** file. For example, if you are using the **Report** item to record the data related to a customer transaction to reserve a rental car, you might name this **Report** item **RentalTrans**.

Tip:

On Voice Portal systems, the Application Report displays this setting for entries in the **Activity** column.

- **Message** - Enter in this field text describing what the purpose of this Report item is at this point in the call flow.

  For example, if you are using the Report item at the beginning of a car rental transaction section of the code, you might have a **Type** of **Start**, a **Name** of **RentalTrans**, and in this field you might then enter: **Customer starts the reservation transaction.**

  The report item writes this string to the log file or database exactly as you enter it here.

Tip:

On Voice Portal systems, the Application Report displays this setting for entries in the **Message** column.

- **Variable** - Select the variable whose value you want to write as an entry in the log file or database.

  If this variable is a complex variable, you must also select a **Variable Field**.

Tip:

On Voice Portal systems, the Application Report does not display this setting, but the value of this variable can be viewed in the Message Details page for the Application Report message. See “Application Report” in the Voice Portal documentation library.

- **Variable Field** - If the Variable you selected is a complex variable, select the variable whose value you want to write as an entry in the log file or database.

Tip:

On Voice Portal systems, the Application Report does not display this setting, but the value of this variable can be viewed in the Message Details page for the Application Report message. See “Application Report” in the Voice Portal documentation library.

Return Event 

Type

Event handler

Available from

- **AppRoot** node

- Form Node

Purpose

Returns an event to the calling application. Can be used to propagate an event caught in a module to the calling application, or may be used to throw a different event back to the calling application.

Behavior

Following is an example for when to use the Return Event item, in collaboration with the Exit item.

1. Catch (event=”myEvent”), Return Event. In this scenario, when the “myEvent” event is caught, the module will exit, returning control back to the calling application, and it will return the “myEvent” that was caught.

   In other words, it propagates or re-throws the event to let the calling application handle the event.

2. Catch (event=”error.runtime”), Return Event, Throw (event=”myReallyBadEvent”, message=”Something happened...”). In this scenario, the application catches an “error.runtime” event and the module will exit returning control back to the calling application.

   Instead of propagating the “error.runtime” event, however, it returns “myReallyBadEvent” instead, with the message “Something happened...”. The calling application can catch the “myReallyBadEvent” event, but does not know that the cause of the event was an “error.runtime” event.

3. Catch (“error.badfetch”), Exit. In this case, the application catches an “error.badfetch” event and then exits the application. Exiting the application will essentially disconnect the caller and terminate the session.

   It does not matter if it is a module or a stand-alone application, the application will end. This is typically used when there is some fatal error with the application or system.

Properties

- **Threshold** - A variable that must have a positive integer value.

  This field is available only when the Throw item is placed in an event handler or a node. This field is not available when the Throw item is placed in a Link item.

  This property is used to disable the Throw item until this value is met. For example, if the Threshold property is set to 3, the system ignores the Throw item the first two times the specified event is encountered. Only the third time that the event is encountered does the system actually throw the event.

Say As 

Type

SSML item

Available from

Any level tab in the Prompt File Editor. For more information about the Prompt File Editor, see Using the Prompt File Editor.

Purpose

At times, the interpretation of a particular word, phrase, or number might depend on the context and the intention for how the word, phrase, or number is to be used. The **Say As** property makes it possible to specify the way in which a designated prompt segment is to be spoken.

For example, the text “Dr.” might be interpreted a couple different ways, depending on the context. When used before a person’s name, it is interpreted as “Doctor.” When used at the end of a street name, it is interpreted as “Drive.” To specify to the speech synthesis engine how this is to be interpreted and spoken, use a setting of **name** for the first case and a setting of **address** for the second case.

To give another example, a seven-digit number, 5551234, might have a variety of interpretations, depending what the number represents in context. This number might represent a telephone number, an amount of money, or a simple number:

- If it is to represent a telephone number, you would select **telephone**. In this case, the speech synthesis engine might speak it as “five five five (pause) one two three four.”
- If it is to represent an amount of money, you would select **currency**. In this case, in U.S. currency, the speech synthesis engine might speak it as “five million, five hundred and fifty-one thousand, two hundred and thirty-four dollars.”
- If it is to represent a simple number, you would select number. In this case, the speech synthesis engine might speak it as “five million, five hundred fifty-one thousand, two hundred thirty-four.”

Behavior

The exact behavior of the setting for the **Say As** item depends on a number of factors.

One factor is the language in which the application is developed. A currency amount, for instance, would be rendered as dollars with the U.S. English language as the default application language. If the application language, however, is U.K. English, the same currency amount would be rendered as pounds.

The default application language also affects which **Interpret-As** formats are supported. Different languages require and support different formats.

Probably the biggest factor that affects the behavior of the **Say As** item is the **Interpret-As** formats that the speech synthesis engine supports. For example, if you use an **Interpret-As** format of **net**, but the speech synthesis engine does not support that format, the system ignores the **Say As** setting. For the **Interpret-As** formats supported by your speech synthesis engine, see the documentation for your speech synthesis engine.

Properties

- **Interpret-As** - Select the format the system is to use to render the text.

Set VDU Fields 

Type

IC Connector item

Available From

Data Node (IC section)

Purpose

Set VDU Fields allows you to set up to 20 vdu variables in one round trip to the VOX.

Behavior

You must first set all the **vdu_cache** values that you want to send. When this node is invoked, the fields indicated will be pulled from the cache and sent over to the VOX in one round trip.

Properties

- **vdu_field1 - vdu_field20** - Indicates the name of the vdu fields to set. You must first add the field to the vdu complex variable, then set the value of the corresponding fields in the **vdu_cache**.

Simple Variable 

Type

Variable item

Available from

Variable Editor only (see Using the Variable Editor)

Purpose

The **Simple Variable** item creates a variable that can contain only one value. This variable can be used as any of the defined variable types. For more information about creating and using simple variables, see Working with Variables.

Behavior

When you create a simple variable in the Variable Editor, Dialog Designer automatically assigns it a default name. You can rename the variable. You can also assign it a default value.

After you create the variable, you must save the application before the variable becomes available for use in nodes or node sub-items.

Properties

- **Name** - Enter in this field the name you want to identify the variable.

  Note:
  This name must follow conventions for naming Java components as described in Naming Java Components.

- **Value** - Enter in this field the default value you want to assign to the variable, if any. If you leave this field blank, the default value is undefined.

Supervised Transfer

See Consultation Transfer.

Text Variable 

Type

Prompt Segment item

Available from

Any level tab in the Prompt File Editor. For more information about the Prompt File Editor, see Using the Prompt File Editor.

Purpose

Text variables have two uses: One use of the **Text Variable** item is to render the value of a variable or variable field as a prompt segment, using speech synthesis (TTS). Another use, in the case of prerecorded audio files, is to play a designated audio file as a prompt segment.

Behavior

When you place a **Text Variable** item in a prompt, it takes the value of the designated variable (or variable field, if it is a complex variable), and:

- If the variable or variable field contains text, the text variable renders the text as spoken output, using TTS.

  Note:
  Instead of using the text variable with a URL to an audio file, it is recommended to use the Phrase Variable type because it includes additional functionality for playing recorded audio files. (The audio file must be of a supported format: G.711 or *.wav formats only.)

For more information about how to use text variables effectively in applications, see Using the Prompt File Editor to Employ Variables.

Properties

- **Variable** - Select the variable you want to render. If the variable you select is a complex variable, you must also select a **Variable Field**.
- **Variable Field** - If the variable in the **Variable** field is a complex variable, you must select the variable field you want to use.

  The contents of this list depend on which variable you selected in the **Variable** property: Only the fields associated with the selected variable are presented in this list.

  Note:
  If the variable you selected in the **Variable** field is a simple variable, this field is inactive.
- **Format** - Select the format you want to use for the variable. Use the following table to help you determine the correct format.

Note:
If your system supports SSML, Avaya recommends that you use the SSML **Say As** option, as that gives you greater control over the TTS output. See Say As.

Format	Description/Comments
text	Reads the contents of the variable or variable field as normal text. If the contents include numbers, reads them as numbers. The exact interpretation of numbers depends on the TTS engine.
filename	Renders the contents of the file named here. If the file is: • A text file, the system renders it to audible speech using TTS. The text file must have a .txt extension. • An audio file, the system plays the file. The audio file must have a .wav extension. Note that these are the only two file formats supported for this option. Note also that the file must reside in the **/data/temp** directory for the application.
digits	Reads the contents of the variable or variable field as separate digits. For example, if the value of the variable is **H14295**, the system reads the value as “H one four two nine five.”
url	Retrieves the file specified in the URL and plays it back. If the file is: • A text file with a .txt extension, the system renders it to audible speech using TTS. • Any other type file, the system assumes it is an audio file and attempts to play it as such. This format requires that the value of the variable or variable field be a valid URL.

Throw 

Type

Form Item

Available from

- **AppRoot** node (see Setting Global Application Properties: AppRoot Node)
- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node

- Record Node
- Menu Node
- Form Node

Purpose

In Dialog Designer, custom events can be created for situations where they are needed. For example, perhaps a custom event is needed to handle caller requests to be transferred to an operator. Or custom events can be used to handle other out-of-the-ordinary situations.

Use the **Throw** item to throw events created using the Event Editor. For more information about creating and using custom events, see Working with Event Types.

Note:
The default VoiceXML events are thrown automatically, as the system responds to various predefined situations. Avaya designed the **Throw** event to only throw those custom events that have been created.

Behavior

Before using the **Throw** item, an event must be created for it to throw. Also, somewhere in the application, within the scope of this item, a Catch (VXML Events) item must be used.

In most cases, a **Throw** item is used to respond to actions or inputs from callers that are outside the scope of what is expected. For example, perhaps a custom event is created in which the caller can request to transfer to an operator from anywhere in the call flow. To do this, create an event called **Xfer2Operator**, for example. For more information about custom events, see Custom Events.

A Link item can be used in the AppRoot node, to monitor and respond to a caller request from anywhere in the call flow. When the caller utters the word or phrase that activates the **Link** item, the **Link** item throws the custom event. At this point, set up a Catch (VXML Events) event handler to instruct the application on what to do with the caller (that is, transfer the caller to an operator, in this case).

Throw items can also be used to handle situations where a certain threshold of other events has been reached and you want the system to respond differently after that point. For example, suppose you want to give callers only three chances to make an acceptable response to a Prompt and Collect node. However, if the No Match and No Input event handlers were treated independently and separately, the caller could have a total of five chances before the appropriate event is thrown.

A **Throw** item can be used in each of the **No Match** and **No Input** event handlers, and then catch it within the same node. By setting the **Threshold** for the **Throw** item to 3, then it does not matter whether the caller has a No Match or No Input situation. After the third event, the **Throw** item is activated, and the call flow moves on to whatever you set the **Catch** for this **Throw** item to do.

When an event is thrown, it is caught by the next higher level element in the application. In an input form, event throws at the field level can be caught at the form level, and an event thrown at the form level can be caught at the application root. You cannot catch an event at the same level that it is thrown. An event thrown at the menu level is caught at the application root level.

Tip:
For an example the **Throw** item being used as described, see the sample application named *Events* included with the Dialog Designer software package.

For more information about creating events, throwing events, and catching events, see Working with Event Types.

Properties

- **Event** - Select the event that you want to throw at this point in the application.

  Note:
  If you do not have any custom events defined, this list is populated only with the built-in events for **cancel**, **exit**, and **help**.

- **Threshold** - A variable that must have a positive integer value.

  Note:
  This field is available only when the **Throw** item is placed in an event handler or a node. This field is not available when the **Throw** item is placed in a **Link** item.

  This property is used to disable the **Throw** item until this value is met. For example, if the **Threshold** property is set to 3, the system ignores the Throw item the first two times the specified event is encountered. Only the third time that the event is encountered does the system actually throw the event.

Trace

Type

Tracking item

Available from

- Announce Node
- Prompt and Collect Node
- Blind Transfer Node
- Bridged Transfer Node
- Record Node
- Disconnect Node

- Menu Node
- Form Node
- Data Node
- Tracking Node

Purpose

The primary use of the **Trace** item is as a debugging tool during application development. You can use the **Trace** item to:

- Check the value of variables or variable fields at different points during application simulation and execution.
- Track application progress, to make sure it is reaching certain points at run time.
- Define different levels of criticality when unexpected or error-type events occur.

Behavior

During simulation or run time, the **Trace** item takes the value of the **Text** field and any selected **Variable** or **Variable Field** and writes the value to a log file. The application locates this log file in the ***ProjectName*/data/log** directory, where ***ProjectName*** represents the name of the speech application project.

To view the log file contents in Dialog Designer, locate the appropriate log file in this directory and double-click it. To view the log file contents in a run-time environment, locate the appropriate log file on the application server, and view it with your text viewer or word processor of choice.

Tip:

The name of the log file for the current day is always **trace.log**. Each time you run a simulation, Dialog Designer appends the log data to the end of this file. At the end of each day, Dialog Designer renames the log file to: **trace.log.*yyyy-mm-dd***, where ***yyyy-mm-dd*** represents the date on which the log file was compiled.

During simulation, Dialog Designer also displays tracing data in the Console view display, and you can review it there.

Note:

Trace log data is written using **log4j** logging technology. Because this is new technology and available information constantly changes, for more information about log4j technology, Avaya recommends that you perform an Internet search with the search term “log4j”.

The **Trace** item requires that you enable it before the system recognizes it during simulation or run time. For this reason, if you use a **Trace** item in a call flow, and if you do not enable tracing output during simulation or run time, the system ignores it and does not write any trace data to the log file.

To enable tracing during simulation:

1. From the **Window** menu, click **Preferences**.

   Dialog Designer displays the Preferences dialog box.

2. In the navigation pane on the left, click the plus [+] symbol next to the **Dialog Designer** entry. The **Dialog Designer** entry expands to show its subentries.

3. In the expanded subentry list, click **Avaya Application Simulator**.

   Dialog Designer displays the preferences options for the Avaya Application Simulator.

4. Enable both of the following options (among others displayed, as desired):

   - **Enable Dialog Designer logging of tracing output**
   - **Enable Application logging of tracing output**

5. These and other Avaya Application Simulator preferences are described Avaya Application Simulator Preferences.

6. Click **OK**.

Because the **Trace** item is considered primarily a development and debugging tool, it is not normally used with a deployed application during run time. The log files that are generated during run time can consume large amounts of drive space. However, they may be useful when testing or troubleshooting an application deployment.

To enable/disable tracing in a run-time environment:

1. Deploy and configure the speech application on the application server.

2. Edit the **ddrt.properties** file in the /data directory. A **ddrt.properties** file is available in the project **/data** directory that controls logging parameters.

   In the development environment, these parameters can be controlled through Avaya Application Simulator Preferences settings. In a deployed system, by editing the **ddrt.properties** file, the next run session will pick up the settings. The configuration settings are described in the following table.

   ddrt.properties file Configurable Parameters

Parameter	Description
localddtrace	Dialog designer built-in tracing. Valid values are enabled or disabled.
localapptrace	Application tracing. Valid values are enabled or disabled.
showvxml	Enables the local **report.log** on platforms (Voice Portal) when they are normally disabled. Valid values are enabled or disabled.

ddrt.properties file Configurable Parameters (continued)

Parameter	Description
localreportlog	Enables the local **report.log** on platforms (Voice Portal) when the local log is normally disabled. Valid values are enabled or disabled.
skeletonreporting	Enables application skeleton reporting. This is reporting that tracks the path a caller takes through an application. Valid values are enabled or disabled.

Properties

- **Priority** - Select the priority level to assign to this **Trace** item.

  The priority level you select is written to the log file as part of the tracing data. This makes it easy to search the log file for specific problems based on the priority of the trace entry.

  The following table lists the priority levels for **Trace** items, shows how they appear in the log file, and provides a brief description of what each level means.

Priority	Log Entry	Description
Fatal	FATAL	A situation in which the application would crash.
Error	ERROR	A situation that would cause the application to generate an error.
Warning	WARN	A situation that would cause the application to generate a warning.
Info	INFO	A situation that would cause the application to generate an informational message.
Debug	DEBUG	A situation that would cause the application to generate debugging data.

- **Text** - Enter in this field any text that you want the system to record as part of the **Trace** item entry in the log file. The system enters the text in the log file verbatim.
- **Variable** - Select the variable whose value you want to record to the log file as part of the **Trace** item entry.

  If this variable is a complex variable, you must also select a **Variable Field**.
- **Variable Field** - If the **Variable** you selected is a complex variable, select the variable field whose value you want to record to the log file as part of the **Trace** item entry.

Transfer Call 

Type

IC connector item

Available from

Data Node only

Purpose

The **Transfer Call** item makes it possible to have the IC system transfer a caller to a destination.

Behavior

Before you can use this or any other IC item, you must enable IC for Dialog Designer. For more information about the ability of Dialog Designer to interact with IC, and to enable IC in your Dialog Designer applications, see About the IC Connector.

Note:
Support for call transfer functions can vary from one IVR system to another. For more information and details about support for transfer functions on your system, see the documentation for your IVR system.

When the speech application encounters the **Transfer Call** item, the application directs the IC system to initiate a call transfer to the destination as assigned in the **Destination** property. See the “Properties” section that follows.

Properties

- **Destination** - Enter in this field the destination to which the IC system is to transfer the call. Valid entry types include:
  - A number that represents:
    - A specific agent’s extension number.
    - A telephone number that is configured to accept and route such incoming calls.
  - An alphanumeric string that represents an agent’s login ID.

    This type of entry transfers the call to the agent associated with this login ID.

Try

Type

Data item

Available from

Data Node

Purpose

To try to execute one or more operation items. If any of the operations beneath the Try throw an exception, the Java exceptions can be caught at runtime.

Behavior

Try/Catch items in the Data Node will *try* to execute items under the Try item. To handle exceptions, add Catch (Exception) items (under the parent Try). For every Try item, you need at least one Catch (Exception).

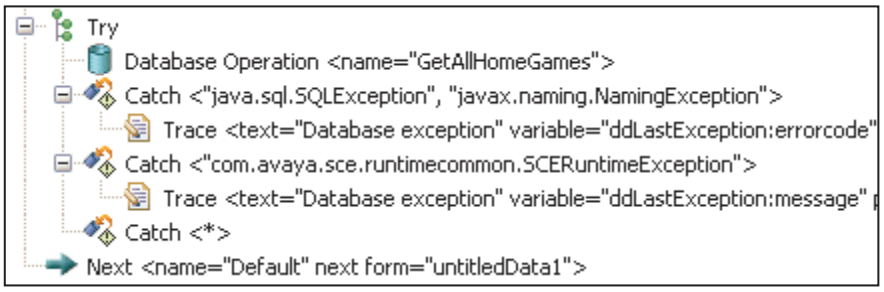
By default, Catch (Exception) will catch *all* exceptions. Catch (Exception) can also be used to catch specific exception types for handling specific errors differently. Comma separated exception types can also be specified for catching different exception types, but handling them in the same way.

List of common exceptions are provided in the Properties view. Currently this list contains:

- SQLException
- IOException
- SCERuntimeException

A new variable called **ddLastException** automatically captures the exception caught by a Catch (Exception) item (includes the following data members: errorcode, message, object, stacktrace, and type).

Following is an example of the Try/Catch mechanism.



TTS 

Type

Prompt Segment item

Available from

Any level tab in the Prompt File Editor. For more information about the Prompt File Editor, see Using the Prompt File Editor.

Purpose

The **TTS** item renders text input as synthesized speech output. This can be valuable when it is impossible or impractical to use prerecorded audio files for prompts.

Behavior

The TTS item takes text entered into the TTS Text field and passes it to the TTS speech synthesis engine. The TTS engine then renders the text input into audible synthesized speech.

To control the pitch, rate, volume, and other aspects of the TTS rendering, use the SSML options. The SSML options include:

- Break
- Emphasis
- Prosody
- Say As
- Voice

Properties

- **TTS Text** - Enter in this field the text that you want rendered as synthesized speech.

Value Operand 

Type

Database Operation Condition Operand

Available From

Database Operation Editor Predicate Tab

Purpose

It is used by the Simple condition item to set up search criteria with a constant value.

Behavior

A Simple condition requires having one Value operand or one Variable operand. After adding a Simple condition item, you need to add the Value operand if you want to search for a pre-determined value.

Properties

- **Value** – put in any constant value in the string format.

Variable Operand 

Type

Database Operation Condition Operand

Available From

Database Operation Editor Predicate Tab

Purpose

It is used by the Simple condition item to set up search criteria with a value that comes from a variable in runtime.

Behavior

A Simple condition requires having one Value operand or one Variable operand. After adding a Simple condition item, you need to add the Variable operand if the value you are searching for comes from a variable in runtime

Properties

- **Name** – choose a variable from the list
- **Field** – choose a field from the variable selected.

Voice 

Type

SSML item

Available from

Any level tab in the Prompt File Editor. For more information about the Prompt File Editor, see Using the Prompt File Editor.

Purpose

Because of cultural or other preferences, you might find it desirable to change the type of voice the speech synthesis engine uses to speak text. The **Voice** item makes it possible to control various aspects of the voice used to render text, including gender and age.

Behavior

Based on the properties you set, the **Voice** item controls the type of voice used to render text. All properties are optional, but if you use the **Voice** item, you must use at least one property. The **Voice** item has four properties:

Note:
The specific effects of these properties might vary from one TTS engine to another.

Properties

- **Gender** - Select the gender you want the voice to have. The choices include: **Male**, **Female**, and **Neutral**.
- **Age** - Enter in this field a non-negative integer.

  This number represents the supposed age of the speaker. So, if you want a childlike voice to render the text, you might set this to **6** or **7**. If you want the voice to sound like an older person, you might set this to **60** or **70**.
- **Variant** - Enter in this field a positive integer.

  This number represents an alternate voice as defined by the speech synthesis engine. Valid values are determined by the speech synthesis engine. So, to use this property, consult the documentation for your speech synthesis engine.
- **Name** - Enter in this field the name that you want to identify the voice.

  This name is, usually, a name of a person. So, for example, a male English voice might be named “John,” and a female Spanish voice might be named “Juanita.”

Note:
Even though Dialog Designer does not enforce it, this name should follow conventions for naming Java components as described in Naming Java Components.

Web Service (Operation) 

Type

Node item

Available from

Data Node

Purpose

The **WebService** item makes it possible to use a predefined Web service operation. The **WebService** item in itself only invokes the Web service operation and tells the call flow where to use it.

Behavior

Before you can use the **WebService** item in the Data node, you must have a Web service operation defined for the application. For more information about defining and using Web service operations, see Working with Web Services.

The way the Web service operation actually behaves in the Data node depends on how the Web service operation is defined. When you place the **WebService** item in the Data node flow, all Dialog Designer does is invoke the selected Web service operation.

Properties

- **Name** - Select the Web service operation you want to invoke.

Appendix B: Project Properties

Many project properties are established at the time an Avaya Dialog Designer speech project is created. Other properties you are prompted to examine and possibly change when you deploy the project for use.

There might be, however, times when you want or need to change various project properties; for instance, to add a database source to the project or to add a language.

This appendix provides information about setting project properties for the Dialog Designer speech project. It contains the following sections:

Setting Project Properties

To set project properties, you must open the **Properties for *ProjectName*** dialog box, where *ProjectName* is the name of the project.

To open the project properties dialog box:

1. Perform one of the following actions:
   - On the **Project** menu, click **Properties**.
   - Right-click the project name in the Navigator view and, from the pop-up context menu, select **Properties**.

   Dialog Designer displays the **Properties for *ProjectName*** dialog box.
2. In the left pane, select one of the following:
   - To set Dialog Designer properties, click **Dialog Designer**.

     See Dialog Designer Project Properties.
   - To set Tomcat properties for the project, click **Tomcat**.

     See Tomcat Properties.

Dialog Designer Project Properties

The **Dialog Designer** properties pane has the following tabs, depending one whether the properties are for a speech application project or call control application project:

- General Tab - View and edit a variety of project properties.
- Speech Tab - *For speech application projects only*. Set ASR and TTS related options.
- Languages Tab - *For speech application projects only.* Configure project language settings.
- Web Descriptor Tab - View and edit various application parameters that govern how the application operates during development and at run time.
- Data Sources Tab - Define and configure various database sources connect with other resources.

General Tab

Use the **General** tab of the **Dialog Designer** properties pane to:

- View general and project meta information.
- Edit selected project information.

To locate and access this tab, see Setting Project Properties.

The **General** tab has the following fields and options:

Dialog Designer Properties—General Tab

Property	Description
Project Properties	
Name	(Display only) This field is the name of the project as assigned when the project was created.
Version	Enter in this field the number to designate this version of the application. By default, Dialog Designer assigns a value of **1.0.0** to this field.

Dialog Designer Properties—General Tab (continued)

Property	Description
Runtime Version	(Display only) This field is the version of the run-time library (`scert.jar`) being used to build and generate the project. **Note:** This version number is usually the same as the version of Dialog Designer you are using, but not necessarily. If, for instance, you have updated to a patch release of Dialog Designer but you did not at the same time update the run-time support files, this version number might differ from the Dialog Designer version number.
Start Page	*For call control application projects only*. Indicates the CCXML start page (for example: **ccxml/start.ccxml**). This is the start page that simulation would use as well. On the Voice Portal platform, you would configure the URL to be like this: **http://.../Myapp/jsp/start.jsp**.
Mode	*For speech application projects only*. This field is currently not used but is reserved for future use.
Type	*For speech application projects only*. This field is currently not used but is reserved for future use.
Icon Properties	
Small	Displays the *small* icon graphic that Dialog Designer uses to represent the speech project in the Call Flow Editor palette (when exporting it as a Dialog Designer reusable module). See Deploying Projects as Dialog Designer Modules. To select an icon different from the default: 1. Click **Browse**. The **Contents of "icons"** dialog box is shown. 2. Select a graphic file to use for the small icon. The graphic selected must be a GIF file, exactly 16 pixels by 16 pixels in size. 3. Click **OK**. If wrong size graphic file is selected, Dialog Designer displays an **Invalid icon** message and retains the currently selected icon. Otherwise, Dialog Designer replaces the former **Small** icon graphic with the new selection. **Note:** By default, Dialog Designer includes two small icon graphic files: **defaultapplicationsm.gif** and **defaultmodulesm.gif**. To use a different icon, create and import a unique file. To do this, See Importing an Icon File.

Dialog Designer Properties—General Tab (continued)

Property	Description
Large	Displays the *large* icon graphic that Dialog Designer uses to represent the speech project in the Call Flow Editor palette (when exporting it as a Dialog Designer reusable module). See Deploying Projects as Dialog Designer Modules. To select an icon different from the default: 1. Click **Browse**. The **Contents of "icons"** dialog box is shown. 2. Select a graphic file to use for the small icon. The graphic selected must be a GIF file, exactly 32 pixels by 32 pixels in size. 3. Click **OK**. If wrong size graphic file is selected, Dialog Designer displays an **Invalid icon** message and retains the currently selected icon. Otherwise, Dialog Designer replaces the former **Large** icon graphic with the new selection. **Note:** By default, Dialog Designer includes two small icon graphic files: **defaultapplicationlg.gif** and **defaultmodulelg.gif**. To use a different icon, create and import a unique file. To do this, See Importing an Icon File.
Meta Information Properties	
Vendor	If a vendor name was entered during project creation, this field displays that name. To change it, enter a new vendor name.
Category	If a category descriptor was entered during project creation, this field displays that descriptor. To change it, enter a newcategory descriptor. **Tip:** If planning to use this project as a reusable module, what is entered in this field determines where in the Call Flow Editor palette that this module will appear. If left blank, by default, the module appears in a group labeled "Modules." If something else is entered, the module is grouped with other modules that have the same category label. This is useful when building a large and complex application that uses several modules that can be grouped together in the Call Flow Editor palette.
Description	If a project description was entered during project creation, this field displays that description. To change it, enter the new project description. **Tip:** If the project is exported as a Dialog Designer reusable module, this description appears as a pop-up message when the cursor hovers over the name of the module in the Call Flow Editor palette.

Importing an Icon File

To import a graphic file to use as an icon for reusable modules:

1. In the Navigator view, right-click the **icons** directory in the project where you want to import the icon graphic.
2. From the pop-up context menu, select **Import...**

   Dialog Designer displays the **Select** page of the **Import** wizard.
3. In the list of import sources, click **File system** and then click **Next**.

   Dialog Designer displays the **File system** page of the **Import** wizard.
4. Perform one of the following actions:
   - Enter the path to the directory that contains the graphics file you want to import.
   - Use the **Browse** button to locate and select the directory that contains the file you want to import.

   Dialog Designer displays in the left pane the directory you selected and in the right pane a list of all files in that directory. Each file is preceded by a check box.
5. Select the file you want to import.

Note:
Make sure that the file or files you select meet the size and format requirements for Dialog Designer icon files. Files must be GIF format, 16 x 16 pixels for small icons or 32 x 32 for large icons. You can select multiple files to import at once.

6. Verify that the **Into folder** field shows the path to the icons directory for your project.

   If not, use the **Browse** button to locate and select the appropriate directory.
7. Click **Finish**.

Speech Tab

Important:
This tab is only displayed for speech application project properties.

Use the **Speech** tab of the **Dialog Designer** properties pane to:

- Set options for speech recognition.
- Enable SSML for Text-to-Speech.

To locate and access this tab, see Setting Project Properties.

The **Speech** tab has the following fields and options:

Dialog Designer Properties—Speech Tab

Setting	Description
Speech Recognition	
Grammar Compatibility	Select the type of ASR server engine to target ASR grammars. Options are: • **IBM** - Grammars are optimized to work with IBM WebSphere or WebSphere Express ASR engines. • **Nuance 8.5** - Grammars are optimized to work with Nuance 8.5. • **Nuance 9.0** (Quantum) - Grammars are optimized to work with Nuance 9.0. • **Nuance OSR** - Grammars are optimized to work with Nuance OSR engines (includes Quantum support). • **SRGS** - (Default) Grammars conform to the SRGS Version 1.0 standard. **Note**: Nuance 9.0 (Quantum) and Nuance OSR use the same format. Dialog Designer, by default, creates grammars to support all ASR engines. The grammars used at run-time is based on what is selected in this field, and ultimately the value of the following variable in the **web.xml** file: **runtime-asr**.
Grammar Caching	Select the grammar caching option to use. Options are: • **none** - No grammar caching is permitted for this application. • **default** - The application uses the IVR system default setting for grammar caching.
Text-to-Speech	
Enable Speech Synthesis Markup Language (SSML) generation in project prompts	Select this check box to enable SSML markers to work in prompts. To disable SSML markers, clear this check box. If selected, when the Avaya Application Simulator is generating the TTS content, it also generates the SSML data to help with the rendering of that content (even in simulation mode during testing). The Microsoft TTS engine used by Dialog Designer, however, is not capable of generating or using SSML data, so it is not apparent during testing that it is being generated.

Languages Tab

Important:
This tab is only displayed for speech application project properties.

Use the **Languages** tab of the **Dialog Designer** properties pane to:

- View information about languages and localization bundles installed in Dialog Designer.
- Add, edit, or delete languages. See Administering Project Languages.
- Install or uninstall localization bundles. See Administering Localization Bundles.
- Install standard phrases. See Using a Localization Bundle's Standard Phrasesets.

To locate and access this tab, see Setting Project Properties.

Web Descriptor Tab

Use the **Web Descriptor** tab to configure various settings that the application will need to run properly on the target platform. This tab has two tabs of its own:

- Application Tab - Provides options related to the way the project is to operate both in the development and the run-time environment.
- Servlets Tab - Provides options related to naming and behavior of the Java servlets that Dialog Designer generates.

To locate and open this tab, see Setting Project Properties.

Application Tab

The **Application** tab provides options related to port and proxy settings, proxy servers (both ASR and TTS), the run-time environment, and caching of grammars.

To change the settings of any options in the **Parameters** area, use the following procedure:

1. Select the parameter you want to change.
2. Click **Edit**.

   Dialog Designer displays the **Edit context parameter** dialog box.

Note: Do not change the **Name** field for any of these parameters.

3. In the **Value** field, enter the new value.
4. (Optional) In the **Description** field, enter a new description for the starting language.
5. Click **OK**.

The following table lists and describes the settings you can make on this tab.

Setting or Option	Possible Values and Descriptions
General	
Use container default	If you want to use the servlet engine settings (that is, Tomcat or WebSphere settings) for the session timeout value, select this check box. If you want to specify your own session timeout setting for this application, clear this check box.
Session timeout	Enter the number of minutes a session can remain inactive on the IVR system, before the system times out and terminates the session. Valid values for this field are integers. A value of 0 (zero) or less means that the session never times out. If the **Use container default** check box is selected, this field is disabled.
Parameters	
sage.startlanguage	The **Value** column displays the language that is set as the starting language for the application. In most cases, it is not a good idea to change this. The default setting for this field is the project language that was selected and set when the project was first created.
sage.ic.throwexceptions	This is a true/false setting (default is true) that allows exceptions to be disabled when generated in an IC operation so they can be analyzed in the data node.

Setting or Option	Possible Values and Descriptions
runtime-ASR	If the application uses an ASR server, enter the type of ASR server engine that the application is to use. This also dictates the type of grammars to be used. Valid values are: • **SRGS** - (Default) Deployment optimizes the application grammars for standard SRGS usage. This means that any ASR server that supports SRGS grammars should be able to work with this application. • **Nuance 8.5** - Deployment optimizes the application grammars to work with Nuance 8.5 ASR servers. • **Nuance 9.0 (Quantum)** – Deployment optimizes the application grammars to work with Nuance 9.0 ASR servers. • **Nuance OSR** - Deployment optimizes the application grammar to work with Nuance OSR ASR servers (includes Quantum support). • **IBM** - Deployment optimizes the application ASR to work with IBM WebSphere Voice servers.
runtime-SSML	Enter in this **Value** field one of the following: • **true** - Enables SSML generation for run time. • **false** - (Default) Disables SSML generation for run time.
runtime-Platform	Enter in this **Value** field the type of IVR platform the application is targeted to run on. Valid values include: • **Voice Portal** - Deployment optimizes the application to run on an Avaya Voice Portal system. • **IR** - Deployment optimizes the application to run on an Avaya IR system. • **Desktop** - (Default) Deployment optimizes the application to run on one of the supported operating system desktops. This option is intended primarily for development. • **Other** - Deployment optimizes the application to run on other systems that support the W3C VoiceXML 2.0 Recommendation.

Setting or Option	Possible Values and Descriptions
grammar-caching	Enter in this **Value** field the setting that determines how the application handles grammar caching. Valid values include: • **none** - No grammar caching is permitted for this application. • **default** - The application uses the IVR system default setting for grammar caching.

Servlets Tab

On the **Servlets** tab, you can manually edit various parameters and settings for the Java servlets that deployment includes. In nearly all cases, you do not need to make changes to these settings.

CAUTION:
Avaya recommends strongly that you do not change these settings except in rare cases, and then only if you are sure of what you are doing. To change these settings is to risk breaking your application.

Data Sources Tab

Use the **Data Sources** tab of the **Dialog Designer** properties pane to:

- Add, edit, and delete JDBC data sources for the project.
- Enable computer telephony integration (CTI). Telephony servers work with the CTI connectors to provide computer telephony integration (CTI).
- Enable the Interaction Center (IC) connector and its components.

See the following sub-tabs for more information:

- Database Tab
- CTI Tab (*For speech application projects only.*)
- IC Tab (*For speech application projects only.*)

Note:
Because Dialog Designer is a Java-based tool, data sources for Dialog Designer speech applications must be JDBC-compliant.

To locate and access this tab, see Setting Project Properties.

Database Tab

Because Dialog Designer is a Java-based tool, data sources for Dialog Designer speech applications must be JDBC-compliant. If JDBC drivers are not installed on your computer, they must be installed to work with data sources. To install these drivers, see the documentation that cam with your database software.

The **Database** tab is located on the **Data Sources** tab of the **Dialog Designer** properties pane in the **Properties for *ProjectName*** dialog box. For more information about the **Properties for *ProjectName*** dialog box, including the procedure to open it, see Setting Project Properties.

Use the **Database** tab for:

- Adding a JDBC Data Source
- Editing a JDBC Data Source
- Deleting a JDBC Data Source
- Creating (or Removing) Failover Data Sources

Adding a JDBC Data Source

To add a data source:

1. Access the **Database** tab located on the **Data Sources** tab of the **Dialog Designer** properties pane in the **Properties for *ProjectName*** dialog box.
2. On the **Database** tab, click **Add** to add a data source.

   Dialog Designer displays the **Create a new datasource** dialog box.
3. In the **Name** field, enter the name Dialog Designer is to use to identify the data source.

   Note that this can be any name you choose. It is used only within Dialog Designer.
4. In the **Driver Class Name** field, enter the name of the driver class to use with this data source. For more information about what to enter in this field, see the documentation for your database software. For example:

   Driver Class Name: oracle.jdbc.OracleDriver

   Connection URL: jdbc:oracle:thin:@148.147.46.68:1521:orcl

Note:

The WAR file for Tomcat contains the context file that is automatically deployed in the Tomcat directory for the application. Tomcat uses this context file to create and manage data source connections. In case you need to modify the data source information such as the driver class and URL without going though the design environment, you need to go directly into this context file.

The file exists in the **<*Tomcat_Home*>\conf\Catalina\localhost\ directory** with the **<*application_name*>.xml** naming. To modify the driver class or URL, locate the Resource element which has the corresponding data source in its dataSourceName attribute, and change the URL or driver attribute of the same element.

5. In the **Connection URL** field, enter the URL to the data source.

   This URL can be either external to the local system or somewhere on the local system. For more information about what to enter in this field, see the documentation for your database software.

6. If the data source requires a user login, in the **Username** field enter the login user ID, and in the **Password** field enter the login password. For more information about what to enter in these fields, see the documentation for your database software.

7. Click **OK**.

Editing a JDBC Data Source

To edit a data source:

Note:

Before you attempt to perform this procedure, verify that you have the appropriate JDBC driver installed and configured. For more information about installing and configuring the JDBC driver, see Database Preferences.

1. Access the **Database** tab located on the **Data Sources** tab of the **Dialog Designer** properties pane in the **Properties for *ProjectName*** dialog box.
2. On the **Database** tab, select the data source to be edited.
3. Click **Edit**.

   Dialog Designer displays the **Edit the datasource** dialog box.

4. Edit the fields as desired.

   The only field you cannot change is the **Name** field, which is display-only.

5. When finished making changes, click **OK**.

Deleting a JDBC Data Source

To delete a data source:

1. Access the **Database** tab located on the **Data Sources** tab of the **Dialog Designer** properties pane in the **Properties for *ProjectName*** dialog box.

2. On the **Database** tab, select the data source to be deleted.
3. Click **Delete**.

Dialog Designer deletes the data source.

Creating (or Removing) Failover Data Sources

To create a list of failover data sources:

1. Access the **Database** tab located on the **Data Sources** tab of the **Dialog Designer** properties pane in the **Properties for *ProjectName*** dialog box.
2. On the **Database** tab, select a data source that will have a failover data source.
3. A **Configure Failover Path** dialog box is displayed. Click **Add** to select a failover data source.
4. A **Data Sources** dialog box is displayed, listing possible data sources that can be selected as failover data sources. Click on one to select it, then click **OK**.

   Note:
   The list of data sources in this dialog is a subset defined in the Database tab.
5. The selected data source is shown in the **Configure Failover Path** dialog box. Click OK, to save the data source's failover designation.

   If desired, more than one data source can be designated as a failover data source. In this case, repeat steps 3 and 4 for as many data sources desired. To give priority to the different data sources that will be used as failovers, click on the Up and Down buttons in the **Configure Failover Path** dialog box as necessary.

   To remove a data source designated as a failover data source, click **Remove**.

CTI Tab

Important:
This tab is only displayed for speech application project properties.

The **CTI** tab is located on the **Data Sources** tab of the **Dialog Designer** properties pane in the **Properties for *ProjectName*** dialog box. For more information about the **Properties for *ProjectName*** dialog box, including the procedure to open it, see Setting Project Properties. For more information about CTI, see About CTI Connectors.

Use the **CTI** tab to enable CTI. Select the **Enable CTI** check box.

IC Tab

Important:
This tab is only displayed for speech application project properties.

Use the **IC** tab to enable or disable the Interaction Center (IC) connector for Dialog Designer. See About the IC Connector.

The **IC** tab is located on the **Data Sources** tab of the **Dialog Designer** properties pane in the **Properties for *ProjectName*** dialog box. For more information about the **Properties for *ProjectName*** dialog box, including the procedure to open it, see Setting Project Properties.

To enable the IC connector, select the **Enable IC Connector and Components** check box.

When you exit the **Properties for *ProjectName*** dialog box, Dialog Designer enables IC.

When you first enable IC, Dialog Designer installs the IC connector software in Tomcat. After that, to get the IC connector to work for the first time, you must restart the Tomcat server engine.

Tomcat Properties

The **Tomcat** property settings should be checked carefully, especially after updating to a new version of Dialog Designer. If you are having trouble updating applications when you update Dialog Designer, check these settings.

The **Tomcat** properties pane is created by a third-party plug-in that Dialog Designer has integrated. When you create a project, Dialog Designer automatically and properly configures the project's Tomcat properties. Dialog Designer does not, however, continue to check or update these properties as you continue to work on and develop the project.

This section is provided primarily as reference, against the eventuality that these settings somehow get changed or corrupted and the project stops running correctly in the simulator.

CAUTION:
To prevent "breaking" your project and causing it to no longer run correctly in the simulator, Avaya recommends that you do not change any of the settings in this pane, unless they have inadvertently been changed by some other means. If you find it necessary to change them, you should change them only to match the settings in this section.

The **Tomcat** properties pane is where project properties are available for how Tomcat is set to work with Dialog Designer projects. There is one tab, the General tab, used to:

- Verify that the project is a Tomcat project.
- Make general settings for how Tomcat is to work with the Dialog Designer project.

To locate and access this tab, see Setting Project Properties.

The **General** tab provides the following options.

Tomcat Properties—General Tab Options

Option	Description
Is a Tomcat Project	Verify that this check box is selected. This check box must be selected for all Dialog Designer projects to work properly.
Context name	Displays and consists of only a slash (/) followed by the project name. For example, if your project is named **MyProject**, this field should display: **/MyProject**
Can update server .xml file	Verify that this check box is selected indicating that Tomcat can update the server .xml file.
Mark this context as reloadable	Verify that this check box is selected so the context is reloadable.
Redirect context logger to Eclipse console	Verify that this check box is cleared, so the context logger is not redirected to the Eclipse console.
Extra information	Optionally, use this field to enter notes or comments regarding the use of Tomcat with this project.
Subdirectory to use as web application root	Do not use this field. Its only content should be a slash (/).

Appendix C: Configuring Preference Settings

Dialog Designer includes a number of preference panels with settings that can be configured uniquely for your Dialog Designer development environment. If these settings are not adjusted, Dialog Designer uses the pre-set default settings.

This appendix provides information about the preference panels specific to using Dialog Designer. It contains the following sections:

For more information about other Eclipse preferences, see the Eclipse documentation.

Accessing Preferences

To set Dialog Designer preferences that affect all Dialog Designer speech projects:

1. On the **Window** menu, click **Preferences**.

   Dialog Designer displays the **Preferences** dialog box.

2. In the navigation pane on the left, select the set of preferences you want to edit.

   Tip:

   To expand a main entry and view submenu items, click the plus (**+**) symbol to the left of a main entry item.

3. Edit the preferences settings. The preferences that are of particular interest to Dialog Designer users include:

- Dialog Designer Preferences
- Avaya Application Simulator Preferences
- Call Control Preferences
- Database Preferences
- Debug Tracing Preferences
- Password Encryption Preferences
- Speech Preferences
- Tomcat Settings

4. To apply the settings without closing the **Preferences** dialog box, click **Apply**.
5. When you are finished setting preferences, to apply the settings and close the **Preferences** dialog box, click **OK**.

Dialog Designer Preferences

The Dialog Designer Preferences panel includes a setting to enable a n HTTP proxy connection, as follows:

- Enable HTTP proxy connection - If a proxy server is required for Internet access, select this check box.

  If a proxy server is not required for Internet access, clear this check box.

  Proxy settings are required when all of the following conditions are true:

  - The system where Dialog Designer is installed is behind a firewall.
  - Access is required to resources that reside outside the firewall for your Dialog Designer speech projects. These resources may include Web services, databases, or other outside resources.
  - Access to these resources requires the use of either an HTTP or HTTPS proxy server.

  When these conditions are true, proxy settings for Dialog Designer must be configured, even if proxy settings are already configured for your Internet browser or email client. If you have a proxy server configured for your Internet browser, use the same proxy settings for Dialog Designer. See Configuring the Dialog Designer Admin (ddadmin) Web Application.

Avaya Application Simulator Preferences

The Avaya Application Simulator Preferences settings, accessible by expanding the Dialog Designer Preferences item, control the way the Avaya Application Simulator (AAS) functions with Dialog Designer applications during simulations. Usually, these settings do not need to be changed.

Avaya Application Simulator Preferences

Setting	Description
Automatically restart tomcat when running an application.	Though not recommended, when checked, this option restarts tomcat before making a simulation call.
Enable Dialog Designer logging of tracing output.	When checked, enables framework tracing, which will log (to the trace.log and the console view) framework debug tracing.
Enable Application logging of tracing output.	When checked, logs any of the user-defined Tracing statements in the callflow (to the trace.log and the console view).
Enable display of generated VXML.	When checked, logs the VXML that is generated by the application (to the trace.log and the console view).
Enable Application Framework reporting.	When checked, logs “breadcrumb” items to the report log (or for VoicePortal to the application report). **Note:** This option is not currently supported in Dialog Designer.
Enable display of the Avaya Application Simulator output in a console window.	When checked, opens a MSDOS console window which shows all of the low level output of the Voice Browser.
Automatically hangup Avaya Application Simulator when call ends.	When checked, a simulation AAS process is terminated when a simulated call ends (disconnect or application exit).
Use external configuration file.	Though not recommended, when checked, allows for further tuning of the personal Voice Browser.
Show AutoVon keys	When checked, adds four more DTMF buttons next to the dial keypad for sending A, B, C, and D AutoVon tones for input.

Avaya Application Simulator Preferences (continued)

Setting	Description
Enable Bargein detection	Avaya Application Simulator will do bargein detection when this is checked. This setting applies to speech applications only.
Tomcat port	Indicates the port that Tomcat will listen on. 8080 is the default. If Tomcat is configured to use a different port with Dialog Designer, then this field needs to be updated.

AVB Settings

Avaya Voice Browser settings control settings related to how the voice browser handles fetches, the maximum number of documents that can be loaded at a given time, and tracing and application logging settings for various components.

AVB Settings

Setting	Description
VoiceXML Properties	
Fetch Timeout	The default timeout to use when fetching resources from the application server. If this time elapses with no response from the application server, the AVB returns a Bad Fetch error.
Voice Browser Properties	
Maximum Documents	The maximum number of documents that can be loaded simultaneously at any given moment. This limitation is mainly there to prevent a recurrence to run unbound.

AVB Settings (continued)

Setting	Description
Tracing and Logging	Configure tracing and application logging settings for the different component areas of the Avaya Voice Browser. Settings can be configured independently for all of the following component areas: • Client • INET • Interpreter • Java Script Interface • Object • Platform • Prompt • Recognition • Telephony
Tracing	The greater the degree of tracing, the larger and more detailed the trace output is. A setting of Finest for some of the fields can generate huge amounts of trace logging output, so it is recommended that you use only the degree of tracing required to obtain the data you need. Try setting the level to Fine first and increase it only if you do not get the data you need.
Application Logging	Determines what messages will be logged by the application. On a live system, these logs are sent back to the platform. During simulation these logs show up in the VXML Log tab. The log levels can be: • None – All application log messages are ignored. • Fatal – Only fatal level messages are displayed. • Error – Fatal and error level messages are displayed. • Warning – Fatal, error, and warning level messages are displayed. • Info – All messages are displayed.

Call Control Preferences

The Call Control Preferences panel, accessible by expanding the Dialog Designer Preferences item, currently provides a control to set the maximum number of conference participants. The **Max Conference Participants** value allows for testing error conditions like “conference full”. Typically, 2 or 3 participants are specified.

Click on the subentry preference, CCXML Templates, to configure CCXML templates for more efficient CCXML code development using reusable code parts.

CCXML Templates

To simplify repetitive CCXML application development, the designer can create reusable CCXML templates. Click Add to create a template, and the following the configurable CCXML template fields can be defined.

New Template Dialog Fields

Field	Description
Name	Name of the template. Make intelligent to the purpose of the template.
Context	Enter a context of the template, in terms of how the template is anticipated to be used in your CCXML application development. Possible values are: ● ccxml_all - a template that applies to all CCXML code. ● ccxml_new - a clean slate general template, for getting started. ● ccxml_base - a clean slate general template, but perhaps of a more general nature. ● ccxml_tag - a template with one or more commonly used CCXML tags. ● ccxml_attribute - a template with one or more commonly used CCXML attributes. ● ccxml_attribute_value - a template with one or more commonly used CCXML attribute values.
Automatically insert option	Indicate whether the template should be automatically inserted into your CCXML code development. Otherwise, leave unchecked.
Description	Additional information to describe the template and its context.

New Template Dialog Fields (continued)

Field	Description
Pattern	Indicate a common reusable pattern or code section for the CCXML template being defined. When done, click **OK**.
Insert Variable	When building your patterns, use the Insert Variable option to quickly insert commonly used variables. Available insertion variables are: • cursor – The cursor position after editing template variables. • current date – Current date. • dollar – Dollar symbol. • encoding – Creating file encodings. • line_selection – Selected lines. • time – Current time. • user name – User name. • work_selection – Selected lines. • year – Current year.

Database Preferences

The Database Preferences panel, accessible by expanding the Dialog Designer Preferences item, provides a place for adding and removing database JDBC drivers that will be used when running Dialog Designer applications.

Adding a JDBC Driver

To add a JDBC data driver:

1. On the **Database** preferences panel, click **Add**. Dialog Designer displays a file manager window for navigating to a .jar JDBC driver file.
2. Select a ***.jar** file, then click **Open**.
3. Dialog Designer displays the Database Preferences panel again. Click **Apply** to save the selected driver as a driver for Dialog Designer.

Removing a JDBC Drivers

To delete a data source:

1. In the **JDBC Drivers** pane, select the data source you want to delete.
2. Click **Remove**.

Debug Tracing Preferences

The Debug Tracing Preferences panel, accessible by expanding the Dialog Designer Preferences item, provides a setting that, if enabled (checked), collects tracing output for the Eclipse IDE development environment. Tracing output is saved to the following file:

***eclipse_home*/DialogDesigner/dialogdesigner.log**

In addition, the **dialogdesigner.properties** file, in the same directory, is a log4j properties file that controls the tracing output.

Password Encryption Preferences

The Password Encryption Preferences panel, accessible by expanding the Dialog Designer Preferences item, provides a password encryption utility for users to create encrypted passwords that can then be copied into the target application's **web.xml** file.

To create an encrypted password:

1. In the Password Preferences panel, enter a password to be encrypted in the **Password** field.
2. Re-enter the same password in the **Confirm password** field.
3. Click **Encrypt**. An encrypted value is displayed in the **Encrypted value** field.
4. Highlight and copy (Ctrl-c) this value to the appropriate place within the target application's **web.xml** file. For example:
   - For a T-Server password change, copy and replace the encrypted password string in the <param-value> for the <param-name> - "cti.tserver".
   - For a database password change, copy and replace the encrypted password string found at the end of the <param-value> for the <param-name> - "<dd.ds.[ds name]>".

Note:
The database connector allows the database password to change without requiring the application to re-deploy.

Speech Preferences

The Speech Preferences panel, accessible by expanding the Dialog Designer Preferences item, provides preferences and settings applicable to speech application editors and capabilities.

On the main Speech Preferences panel page, to include the timestamp during code generation, check the **Generate timestamp during code generation** option (which is checked by default).

When the Speech Preferences item is expanded in the Preferences tree, the following additional Preferences pages are available, as described in the following sections:

- Call Flow Editor Preferences
- Languages Preferences
- Phrase Preferences
- Prompt Preferences
- Simulators Preferences

Call Flow Editor Preferences

The Call Flow Editor Preferences panel, accessible by expanding the Dialog Designer Preferences item, and then the Speech Preferences item, provides control color mappings for elements used in the Call Flow Editor, warning and error settings related to call flow design activity, and backup management, including enabling and disabling of call flow design backups, and how many.

Call Flow Editor Preferences

Setting	Description
General Settings	
Validate callflow on save	Check this option to validate the call flow on each save of an application.
Organize imports after copying Java code	This option is enabled by default. It allows for a slowing down of the copy and paste function. Advanced users may want to disable this option for better performance. **Note**: May require user interaction if Eclipse is not able to resolve the import. To organize imports manually, open the Java class (**Source > Organize Imports** or Ctrl-Shift-O).
Automatically close editor tab when opening new tab	When this is checked, the Call Flow Editor tab is closed when opening a new tab in your Dialog Design work area. This option is purely preferential, based on a designer's work styles.
Display node location coordinates	Check this option to display the node location coordinates when displaying the call flow in the Call Flow Editor.

Call Flow Editor Preferences (continued)

Setting	Description
Colors Settings	
Colors	Allows for the configuration of color coding to the various elements used in call flows. By clicking on an element, then clicking on the Color button, a color picker is displayed where a color can be mapped for each element. The following elements can have colors assigned to them: ● Bookmarks ● Bookmark Text ● Labels ● Label Text ● Selected Connection
Warnings and Errors Settings	
Warnings and Errors	Select from (enable, or disable) the following warning and error conditions that are desired to be logged to the Problems pane when working with call flows: ● Display code generation failure errors. ● Display paste errors or warnings. ● Warn when renaming call flow nodes. ● Warn when renaming call flow items that will trigger variables to be renamed. ● Warn before deleting items that will trigger variables to be deleted.
Backups The Call Flow Editor by default maintains a backup of the flow document, in the event that the main flow is corrupted. Prior to saving the contents of the main flow, the old main flow file is copied to a backup folder using a “round-robin” file numbering system to maintain multiple backups. The backup files are numbered (for example, main.flow.1, mainflow.2, and so on) but once the maximum number of backups is reached, the numbering sequence starts over.	
Enable backups	Check this checkbox to enable backups of call flows. By default, this option is enabled.
Number of backups to keep	If backups are enabled, indicate how many backups should be kept for each call flow. By default, 2 backups are kept at any given time. The maximum number of backups is 10.

Languages Preferences

The Languages Preferences panel, accessible by expanding the Dialog Designer Preferences item, and then the Speech Preferences item, provides settings to define what languages Dialog Designer recognizes and makes available for use in speech applications. There are three language types that can be added to Dialog Designer for use with applications, as described in the following sections:

- Automated Speech Recognition (ASR) Languages
- Text-to-Speech (TTS) Languages
- Audio Localization Packages

To add a language preference, click **Add** associated with the language or language bundle to be added, then follow the applicable steps described.

ASR or TTS Language:	***Language Bundle***:
1. Dialog Designer displays the **Add <language type> Language** dialog box. 2. Enter the language code for the language to be added. 3. The code must be in ***ll-cc*** format, where *ll* represents a two-character language code, and *cc* represents a two-character country code. For example, **en-us** for U.S. English or **en-uk** for UK English. 4. The code must match a language code installed on the ASR server/TTS server, or the system cannot process requests that use the language. 5. Click **OK**. 6. Dialog Designer adds the new language to the list of ASR/TTS languages.	1. Dialog Designer displays the **Browse for localization package** dialog box. 2. Locate and select the JAR file containing the localization bundle you want to add and click **Open**. 3. Dialog Designer automatically adds the localization bundle to the workbench environment. The new localization bundle is displayed in the **Audio Localization Packages** list. 4. Click **OK**. 5. Dialog Designer adds the new language bundle.

To delete a language preference, select it and click **Delete**. When prompted to confirm the deletion, click **OK**. When warned that removing the language might invalidate projects, click **Yes** if you still want to delete it.

Phrase Preferences

The Password Preferences panel, accessible by expanding the Dialog Designer Preferences item, and then the Speech Preferences item, provides default settings for recording phrases in Dialog Designer. Edit the fields in this panel, as described in the following bullets:

- **Default audio extension** – Determines the audio recording file format to use when saving phrases in Dialog Designer, by default. Select either:
  - wav – Usually used on Windows-based development environments
  - au – More commonly used on Sun and UNIX development environments.
- **Audio recorder** – Sampling settings for audio recordings. Select the sampling rate (8KHz, 11KHz, 16KHz, 22KHz, 44KHz) and sampling size (8-bit or 16-bit), as appropriate.

Important:
Do not change the Audio Recorder settings. Currently, Dialog Designer only works with the default settings.

Prompt Preferences

The Prompt Preferences panel, accessible by expanding the Dialog Designer Preferences item, and then the Speech Preferences item, provides configurable default settings for a number of prompt management capabilities, as follows.

Prompt Preferences

Setting	Description
Default Bargein	Indicates whether bargein (allowing callers to barge in, or interrupt, prompts) is enabled or not. Options are: • **true** - enabled. This is the default. • **false** - disabled
Default Barge Type	Type of action that triggers barge-in, if enabled for the prompt. Options are: • **speech** - Any spoken response triggers barge-in and stops the prompt. This is the default. • **hotword** - Only a complete match of an active grammar triggers barge-in and stops the prompt.

Prompt Preferences (continued)

Setting	Description
Default Play Order	Determines the order in which prompt levels are played when the prompt is repeated. Options include: • **standard** (default) • **first** • **random** • **sequential** See About Play Order for Prompt Levels.
Default Time Out	Number of seconds or milliseconds that the system must wait for a response from the caller, after the prompt is finished playing, and before the system throws a No Input event. 8 seconds is the default.
Standard Phrases	Click this option to show any standard phrases that have been installed to be available for use in the Prompt File Editor. See Accessing the Prompt File Editor.

Simulators Preferences

The Simulators Preferences page, accessible by expanding the Dialog Designer Preferences item, and then the Speech Preferences item, provides options to configure a number of VOX servers on the VOX simulator. A VOX can operate in two modes:

- it can initiate the connection to the IC connector
- it can allow the IC connector to connect to it

For each VOX server, the following configurations are required:

- Name of the VOX server
- Host IP address (or localhost)
- Port on which it communicates with the IC connector (by default, 3000)
- Extensions
- Mode (either IC Connector --> VOX Simulator, or VOX Simulator -> IC Connector)

The IC Connector on the local machine is automatically picking up on these settings. If you for example configure one VOX simulator to initiate a connection to the IC Connector on port 3000, then when you start the local Tomcat server up its IC Connector will automatically come up and listen for a VOX connection on port 3000.

Note that by setting the host, it is also possible to connect to a real VOX server while your application is run in the Avaya Applications Simulator.

Note:
An instance of the VOX simulator will only be started for each VOX that has host set to localhost. These settings are in effect on the local machine's IC connector the next time that Tomcat is started.

To add a VOX simulator:

1. Click **Add** and a VOX server instance is added with default settings.
2. Click again to add another, though a Warning message also appears stating that "All VOXs must have a unique name."

   Click on the name of the second VOX server added to edit the name.
3. Click on any other default VOX server settings to edit them. For Mode, a drop down option is offered, to select either IC Connector --> VOX Simulator, or VOX Simulator -> IC Connector.

To delete a VOX simulator, click on a field of an existing VOX server instance, then click **Delete**. The VOX server instance is deleted.

Note:
By setting the host, it is also possible to connect to a real VOX server while an application is run in the Avaya Applications Simulator.

Tomcat Settings

The Tomcat Preferences panel provides settings that determine how Dialog Designer works with the Apache Tomcat servlet engine during simulations.

To set the Tomcat preferences to work properly with Dialog Designer, configure the following settings:

- **Tomcat home** - path to the Tomcat software, enter it in the field.

  Or, click **Browse** to navigate to the directory where Tomcat is installed.

  If you used the default installation location, the path is:

  C:\Program Files\Apache Software Foundation\Tomcat 5.0

  where *C:* is the drive letter of your primary drive.
- **Context files** - path to the Contexts directory. The default installation location is:

  C:\Program Files\Apache Software Foundation\Tomcat 5.0\conf\Catalina\localhost

  where *C:* is the drive letter of your primary drive.

Note:
The correct path must take you to the **...\Catalina\localhost** directory.

Appendix D: System Variables

Every speech application project you create with Avaya Dialog Designer has a set of complex variables that are automatically created with the project. These variables are automatically built-in to every speech project you create in Dialog Designer. These are known as *system variables*. These mostly read-only variables can be put to a wide variety of uses in your speech application projects.

For more information about these and about variables in general, see Working with Variables.

This appendix contains the following sections:

System Variable Fields and Properties

The following table lists and describes the fields and properties for the system variables.

System Variable Fields and Properties

Variable	Associated Fields	Descriptions/Comments
date - This variable gets data on the current date from the system.		
	audio	Returns the current date. This field is specifically designed and optimized to be used in an audio variable. Therefore, to use this field, you must have the localization bundle and standard phrases installed. See Audio Variable. When you use this field in a prompt, you can specify additional properties that govern how the information is presented to the caller. See Audio Field Properties.
	dayofmonth	Returns the number of the current day. For example, if the current date is May 23, this field returns the value **23**.
	dayofweek	Returns the name of the current day of the week, for example, **Tuesday**.

System Variable Fields and Properties (continued)

Variable	Associated Fields	Descriptions/Comments
	dayinweeknum	Returns the number of the day in the week, where Sunday is 0 and Saturday is 6.
	dayofweeknum	Returns the current day of the week as an integer, for example, Sunday is **1** ... Saturday is **7**.
	dayofyear	Returns the number of the current day in the year. For example, if the current date is May 23, in a non-leap year, this field returns the value **143**.
	month	Returns the name of the current month, for example, **May**.
	monthinyear	Returns the number of the current month in the year. For example, if the current month is May, this field returns the value **05**.
	year	Returns the current year, for example, **2005**.
redirectinfo - This variable gets data from the session connection redirect variable. This variable is an array representing the connection redirection paths.		
	presentationinfo	Returns the value of the **pi** (presentation information) property for the **session.connection.redirect** variable for each element of the array.
	screeninginfo	Returns the value of the **si** (screening information) property for the **session.connection.redirect** variable for each element of the array.
	uri	Returns the value of the **uri** property for the **session.connection.redirect** variable for each element of the array.
session - This variable takes the values for its fields from the call session information generated at run time.		
	aai	Returns the value of the session variable **session.connection.aai**. This variable contains application-to-application information that is passed during connection setup.
	ani	Returns the ANI, or the number that the caller is calling from.

System Variable Fields and Properties (continued)

Variable	Associated Fields	Descriptions/Comments
session (continued)	**calltag**	Returns a string, which contains the following elements: • Host name - Name of the Avaya IR system from which the call session data was obtained • Channel number - Number of the IR system port where the call was received • Date and time the IR system received the call For example, a typical return value for this field might be something like: IR11 15 Wed Aug 3 11:51:54 2005 **Note:** This field is specific to Avaya IR 1.3 systems.
	channel	Returns the number of the port on which the call resides.
	currentlanguage	Returns the current default language as defined by the application.
	dnis	Returns the DNIS, or the number that the caller dialed.
	lasterror	When an IC operation completes, error information is returned from the VOX in this variable. If the operation succeeded, the lastError field will be empty. In conjunction with the web parameter **sage.ic.throwexceptions**, the application can be disabled from throwing runtime exceptions for IC errors by setting this value to false. This requires that the developer always checks for a value in the **lastError** field, but provides the flexibility of doing so within the data node instead of their error handler. Regardless of the value of sage.ic.throwexceptions, **session.lastError** will always be populated after an IC call.
	protocolname	Returns the value of the session variable **session.connection.protocol.name**. This variable is the name of the connection protocol.
	protocolversion	Returns the value of the session variable **session.connection.protocol.version**. This variable is the version of the connection protocol.
	sessionid	Returns the session ID number assigned to the call.

System Variable Fields and Properties (continued)

Variable	Associated Fields	Descriptions/Comments
session (continued)	**sessionlabel**	Contains any data you want to assign to it. This is the only field of the session variable to which you can assign a value. Assign the value using the **Value** field in the **Avaya Properties** view. **Note:** This field is specific to Avaya Voice Portal. It was designed to be used as an application report filter.
	uui	Returns the contents of the **uui** (user-to-user information) variable field.
	vpcalledextension	Returns the number of the station on the Voice Portal system **Note:** This field is specific to Avaya Voice Portal. Its value is obtained from the Voice Portal Media Processing Platform (MPP).
	vpconverseondata	Contains the DTMF digit data from an Avaya Call Center vector program. This data is used to initiate a session with the Dialog Designer application. **Note:** This field is specific to Avaya Voice Portal. Its value is obtained from the Voice Portal Media Processing Platform (MPP). For more information about converse-on transactions, see your Voice Portal documentation.
	vpcoveragereason	Returns the reason the call is being transferred to the Voice Portal system. Possible values include: • noanswer • busy **Note:** This field is specific to Avaya Voice Portal. Its value is obtained from the Voice Portal Media Processing Platform (MPP).
	vpcoveragetype	This field is reserved for future use. **Note:** This field is specific to Avaya Voice Portal. Its value is obtained from the Voice Portal Media Processing Platform (MPP).

System Variable Fields and Properties (continued)

Variable	Associated Fields	Descriptions/Comments
session (continued)	**vprdnis**	Returns the original number that the caller dialed, when the caller has been transferred or redirected. For example, if the caller dialed extension 1234, but the party at extension 1234 did not answer, the system might be set to automatically transfer the call to extension 9000. In this example, the **dnis** field would be set to 9000, and this field, the **vprdnis** field would be set to 1234. **Note:** This field is specific to Avaya Voice Portal. Its value is obtained from the Voice Portal Media Processing Platform (MPP).
	vpreporturl	Returns the URL of the Web service to which the system writes the application report data. **Note:** This field is specific to Avaya Voice Portal. Its value is obtained from the Voice Portal Media Processing Platform (MPP).
time - This variable gets data on the current time from the system.		
	timezone	Returns the time zone in which the system is configured, as a number relative to "Zulu" or Greenwich Mean Time. For example, if the system taking the call is configured for Eastern Daylight Savings Time, then this field returns a value of **-0400**.
	minute	Returns the current minute past the hour on the system clock. For example, if the current time is 5:32 p.m., this field returns **32**.
	audio	Returns the current time. This field is specifically designed and optimized to be used in an audio variable. Therefore, to use this field, you must have the localization bundle and standard phrases installed. For more information about using audio variables, see Audio Variable. When you use this field in a prompt, you can specify additional properties that govern how the information is presented to the caller. See Audio Field Properties.
	millisecond	Returns the current millisecond of the system clock at the time the request is made.

System Variable Fields and Properties (continued)

Variable	Associated Fields	Descriptions/Comments
	hour	Returns the current hour of the system clock, in a 24-hour format. For example, if the current time is 5:32 p.m., this field returns **17**.
	second	Returns the current second of the minute past the hour on the system clock. For example, if the current time is 5:32:45 p.m., this field returns **45**.

Variable Formats in Localization Bundles

All language localization bundles are required to support certain variable data formats. These formats, in turn, are the basis of the related language localization bundle formats.

For example, all language localization bundles must support the **date** data format:

YYYY[**-**]***MM***[**-**]***DD***

where *YYYY* is the four-digit year, *MM* is the two-digit month, *DD* is the two-digit day, and the hyphens are optional.

If you want to use the standard localization bundle formats in custom variables that you want to use as audio variables, you must ensure that your custom variables follow the standard formats as described in the following table. Then you can use the localization bundle formats to render the data.

For example, suppose you want to query a database for all customer transactions that took place in the previous five days. When you query the database, you can use the **Operation** item of a Data node to convert the date field for each record to the format ***YYYYMMDD***. You can then use audio variables to convert this ***YYYYMMDD*** value into a set of audio clips to speak the date.

The following table lists and describes the standard formats that all language localization bundles must support. The **Data input format** column provides information about the formats that you can use in your own custom variables.

Note:

Localization bundles can, and often do, include additional variable formats that are not listed in this table. These formats are only the ones required of *all* localization bundles. For more information about any additional variable formats that might be part of a particular localization bundle, see the supplemental documentation that came with that bundle.

Localization Bundles Data Input Formats

Format	Description	Data Input Format [1]
Date	Formats a string into a set of audio clips to render the string as an audio variable in a date format, as defined by the localization bundle.	***Type***: String ***Input format***: ***YYYY***[**-**]***MM***[**-**]***DD*** ***Examples***: **2005-05-25** or **20050525**
Time	Formats a string into a set of audio clips to render the string as an audio variable in a time format, as defined by the localization bundle.	***Type***: String ***Input format***: ***HH***[**:**]***MM***[**:**][***SS***] ***Examples***: **03:15:45** or **03:15** or **031545**
Number	Formats a number-digit string into a set of audio clips to render the number as an audio variable in a number format, as defined by the localization bundle. Commas and periods are used as delimiters for this format. The localization bundle treats the rightmost delimiter as the decimal indicator. **Note:** Localization bundles, in some cases, can define numbers to be treated as integers, in which decimal and fractional parts are ignored. Make sure you know which number formats in your localization bundle support the use of decimals and fractional parts.	***Type***: String ***Input format***: [**-**]*#*[**,** or **.**]*###*[**,** or **.**]*##* ***Examples***: **102473** or **15,588.345** or **10.123,52**

Localization Bundles Data Input Formats (continued)

Format	Description	Data Input Format [1]
Currency	Formats a number-digit string into a set of audio clips to render the number as an audio variable in a currency format, as defined by the localization bundle. Commas and periods are used as delimiters for this format. The localization bundle treats the rightmost delimiter as the decimal indicator.	***Type***: String ***Input format***: [**-**]*#*[**,** or **.**]*###*[**,** or **.**]*##* ***Examples***: **102473** or **15,588.345** or **10.123,52**
Character	Formats a string into a set of audio clips to render the string character by character. At a minimum, localization bundles support: ● Letters A through Z ● Numbers 0 through 9 ● Star symbol (*) ● Hash symbol (#) ● Comma (,) ● Space With this format, words are spelled out and numbers are read out one digit at a time. Localization bundles can also support additional characters, including non-Latin characters. For details about which additional characters might be supported by your localization bundle, see the supplemental documentation that came with the bundle. Localization bundles ignore any characters that are not supported by that bundle.	***Type***: String ***Input format***: ***xxxxxx...*** ***Examples***: ● **what** (rendered as "double-you aitch ay tee") ● **10247** (rendered as "one zero two four seven") ● **401K** (rendered as "four oh one kay")

1. Characters in brackets [ ] are optional.

Audio Field Properties

Both the **date** and **time** system variables contain fields labeled **audio**. These fields are specifically designed and optimized to be used with audio variables in prompt segments. Therefore, to use these fields, a localization bundle and standard phrases must be installed. For more information about using audio variables, see Audio Variable.

When these fields are used in prompts, additional properties that govern how the information is presented to the caller can be specified. These properties are dependent on the **Format** property selected:

- When using the **audio** field of the **date** system variable in a prompt, use **Date** for the **Format** property.
- When using the **audio** field of the **time** system variable in a prompt, use **Time** for the **Format** property.

Assuming the correct **Format** property is selected, additional properties presented in the **Avaya Properties** view depend on which language localization bundle is installed and being used in Dialog Designer. For details regarding these additional properties, see the supplemental documentation included with the localization bundle.

Appendix E: Creating Scripts for Testing

Dialog Designer uses the Avaya Voice Browser (AVB) to simulate speech applications for testing and debugging purposes. As described in Application Testing by Simulation, you can use the AVB to simulate a wide variety of scenarios. With the AVB, you can manually enter various inputs. You can also use scripts to automate the entry of inputs.

To work with the AVB, simulation scripts must be in the form of XML files. You can save and store them wherever you want, as long as the AVB is able to find and use them.

See the following sections for details on working with scripts:

- Using Scripts to Simulate Caller Responses
- Using Scripts to Simulate IC and CTI Connectors

Avaya has included sample scripts for both these types within the respective sections. Copy and modify them for your own use as needed.

Using Scripts to Simulate Caller Responses

In cases where you do not want to manually simulate responses made by callers, whether they are ASR, DTMF, or action responses, you can use an XML script to automate the responses. You might do this because you do not have a microphone attached to your computer, because you do not want to disturb co-workers, or for other reasons.

Note that, if you do not enable scripting in the Avaya Voice Browser (AVB), or if no configuration is defined for a particular command, the AVB performs the default action for that command. See Summary of Connector Commands.

To use a script to simulate caller responses:

1. Create the caller response script.

   See Creating a Response Script.

2. Save the script file to the desired location, and make a note of where you saved it.
3. On the Avaya Voice Browser **Script** tab, select the check box labeled **Enable script file processing**.

Tip:

To prevent the AVB from trying to recognize any inputs from a microphone, Avaya recommends that you turn off your microphone when using a script to simulate audio input.

4. Perform one of the following actions:
   - In the **Script file name** field, enter the full path to the script. An example of this is:

     C:\temp\simulationScript.xml
   - Use the **Browse** button to navigate to and select the script file.
5. Run the application simulation.

Creating a Response Script

To work correctly with the Avaya Voice Browser (AVB), the response script must be in the form of an XML file. You can use any good text editor to create the file.

When creating your script, use the following guidelines:

Note:
For a sample script that illustrates these guidelines, see Sample Response Script.

- The response script must begin with the following code as the first line of the file:

  ```
  <?xml version="1.0" encoding ="UTF-8"?>
  ```

- Each response that you want to simulate must adhere to one of the following forms:
  - For ASR recognition responses only:

    ```
    <command name="name" type="type">
       <item value="value1"/>
       <item value="optionalValue2"/>
    </command>
    ```

    Note:
    You can have one or multiple **item** entries for each ASR recognition response. If you want to simulate only a single response, use only one **item** entry. If you want to simulate multiple responses to a single ASR attempt, you can add as many **item** entries as you need.
  - For all other responses:

    ```
    <command name="name" type="type" value="value" />
    ```

    where:
    - *name* is the type of response you want to simulate.

      Valid types include:
      - **rec** - Simulates the results of an ASR or DTMF recognition attempt.
      - **tel** - Simulates the response of the telephony system to various situations, mostly errors.

  - **record** - Simulates the results of a caller recording attempt.
  - **quit** - Terminates the script and the simulation.
- *type* is the specific type of response. The valid values for this field depend on what you have in the **name** field.
- *value* is the actual content, or value, of the response returned. The valid values for this field depend on what you have in the **name** field.

- For each response that the application expects during the course of the simulation, you must provide a simulated response. During simulation, the AVB uses each command entry in turn as it encounters nodes that require a response.

  If you want to simulate all responses for the entire application, therefore, you must have a separate command entry for each expected response.

  Tip:
  Avaya Voice Browser response scripts are useful when working on long and complex applications. An excellent way to use scripts is to enter only as many responses as needed to get to the point in the application that is currently being worked on.

  For example, suppose an application prompts the caller for twelve different responses. You have completed the application up to the sixth prompt and are working on the seventh and eighth prompts. A script can be created that provides responses for only the first six prompts. In other words, the Avaya Voice Browser uses the scripted responses for the first six prompts and then goes into normal interactive mode so that the seventh and eighth prompts can be tested manually.

- The following table lists and describes the valid values for each of the fields in these formats:

Valid Values for Use in Scripts

Name	Type	Value	Comments
rec - Recognition responses			
	asr - spoken responses	Any expected spoken response.	Valid values include any item in an active grammar.
	dtmf - touchtone key presses	One or more key presses	Valid values include numbers, star (*), and pound sign (#) characters.
	err - error	**nomatch**	Simulates a situation in which no caller response matches an expected response.
		noinput	Simulates a situation in which the caller did not respond at all.

Valid Values for Use in Scripts (continued)

Name	Type	Value	Comments
tel - Telephony system responses			
	hangup	n/a	No **value** is required, because this type simulates a caller hanging up. You can either leave this field blank or leave it out entirely.
	err - error	**trans_noanswer**	Simulates an attempted transfer in which the destination number did not answer.
		trans_busy	Simulates an attempted transfer in which the destination number was busy.
		trans_netbusy	Simulates an attempted transfer in which the network was not able to reach the destination because all network lines were busy.
		trans_fardiscon	Simulates a successful call transfer in which the party at the other end hung up.
		trans_netdiscon	Simulates a successful call transfer in which the network for some reason disconnected the call prematurely.
		trans_maxdiscon	Simulates a successful call transfer which was terminated by the system because the length of the call exceeded the maximum allowable amount of time.
record - Responses recorded by the system **Note:** This option does not actually record any responses.			
.	**max** - maximum record time	n/a	Simulates a situation in which the caller exceeded the maximum allowable length of time for a recording and the system terminated the recording. You can either leave the **value** field blank or leave it out entirely.

Valid Values for Use in Scripts (continued)

Name	Type	Value	Comments
	done	n/a	Simulates a situation in which the caller completed the recording and signaled that it was complete. You can either leave the **value** field blank or leave it out entirely.
	err - error	Any positive integer	Simulates a situation in which there was an error in the recording. The value field contains a numeric code that identifies the cause of the error.
	dtmf - touchtone key press	One key press	Simulates the key the caller pressed to terminate the recording.
quit			
	n/a	n/a	Terminates the script and the simulation. Use this option only when you want the script to terminate the simulation. When you use this option, you can either leave the **type** and **value** fields blank or leave them out entirely.

Sample Response Script

The following script is a sample of an caller response script. You can use this sample script as a model to create your own scripts.

The following script contains five command entries that correspond to five prompts in the application call flow:

- The first prompt asks what kind of car the caller would like to buy, gas-powered, electric, or a hybrid.
- The second prompt asks what color vehicle the caller would like to rent.
- The third prompt asks what color the caller would like for the vehicle interior.
- The fourth prompt asks if the caller would like to contact a sales representative.
- The last prompt asks the caller to press one to speak to a sales representative or two to leave a message for a sales representative.

The following response script simulates the appropriate responses to these prompts:

```
<?xml version="1.0" encoding ="UTF-8"?>
<callScript>
   <command name="rec" type="asr">
      <item value="hybrid"/>
   </command>
   <command name="rec" type="asr">
      <item value="blue"/>
   </command>
   <command name="rec" type="asr">
      <item value="tan"/>
   </command>
   <command name="rec" type="asr">
      <item value="yes"/>
   </command>
   <command name="rec" type="dtmf" value="1"/>
</AASScript>
```

Using Scripts to Simulate IC and CTI Connectors

When you use Interaction Center (IC) or Computer Telephony Integration (CTI) connectors in your Dialog Designer applications, you can simulate the actions and functions of those connectors with a connector script. A connector script can make it possible to test your application without having to connect to a live IC or CTI system. For more information about IC and CTI connectors, see Appendix G: CTI and IC Connectors.

To facilitate a useful simulation of the application, you can script the behavior of the connector simulator to respond with a specific action based on scripted input. For example, if you want to test your CTI application to make sure the system responds properly when it encounters an error condition, you can write and use a script to simulate errors with CTI connectors. Or you might want to test the behavior of the IC connector to verify that it does what you intend.

Note that, if you do not enable scripting in the connector simulator, or if no configuration is defined for a particular command, the AVB performs the default action for that command. In most cases, that means the command executes successfully. For more information about default actions, see Summary of Connector Commands.

To use a script to simulate IC and CTI connectors:

1. Create the connector script.

   For more information, including the procedure to do this and a sample script, see Creating a Connector Script.
2. Save the script file to the desired location, and make a note of where you saved it.
3. On the Avaya Voice Browser **Script** tab, select the check box labeled **Enable connector simulation file processing**.
4. Perform one of the following actions:
   - In the **Simulation file name** field, enter the full path to the script. An example of this is:

     ```
     C:\temp\simulation.xml
     ```
   - Use the **Browse** button to navigate to and select the script file.
5. Run the application simulation.

Creating a Connector Script

To work correctly with the Avaya Voice Browser (AVB), the connector script must be in the form of an XML file. You can use any good text editor to create the file.

This section includes the following topics:

- Connector Script Guidelines
- Sample Connector Scripts
- Summary of Connector Commands

Connector Script Guidelines

When creating your script, use the following guidelines:

Note:
For sample scripts that illustrates these guidelines and that you can use as a model, or starting point, for your own scripts, see Sample Connector Scripts.

- The script must begin with the following code as the first line of the file:

  ```
  <?xml version="1.0" encoding ="UTF-8"?>
  ```
- The second and last lines of the file must read as follows:

  ```
  <simulator>
  ```

  ```
  </simulator>
  ```

- Nested between these two tags, you must identify the connector for which the script works, using the following format:

  <connector name="connectorType"

  </connector>

  where *connectorType* represents the type of connector for which this part of the script works. Valid values for this field include:

  – **IC**

  – **CTI**

  Note:
  If required, you can include one *connector* entry for each connector type between the *simulator* tags. That is, you can have one section of your script, between *connector* tags for each type of connector, both IC and CTI. You should not, however, include multiple connector tags for a single connector type within the same script. If you do, the simulator acts upon the first one and ignores any other connector configurations for that connector type.

- Nested between each set of *connector* tags, you configure the commands you want the script to support for that connector simulator.

  The *command* tag set uses the following format:

  <command name="*commandName*">

  </command>

  where *commandName* is the command you want to configure.

  For more information about the commands for each connector type, see Summary of Connector Commands.

  You can include multiple *command* entries between each set of *connector* tags.

- Within each *command* entry, you must have one or more sets of configurations. A configuration is a block of XML code in which you define the behavior of the connector simulator for that command. Each configuration is contained within a set of configuration tags that use the following format:

  <configuration>

  </configuration>

- Each *configuration* is made up of sets of *comparisons* and *actions*:

  – A *comparison* instructs the simulator to compare an input received during the call with a certain value. If the comparison evaluates as true, the simulator is to take the corresponding action or actions.

  A *comparison* uses the following format:

  <compare type="*type*" field="*field*" value="*value*" />

  where:

- *type* represents the type of comparison the simulator is to make. Possible values include:
  - **EQUAL** - Defines a field and a value to compare. With this type, both the **field** and **value** elements are required. You can use the asterisk (*****) as a wildcard for the **value** element.
  - **ALL** - Instructs the simulator to take the action in every case. With this type, you do not use either the ***field*** or the ***value*** element.
- *field* is a predefined value that must correspond with the command being configured. For more information, including the fields that are valid for each command, see Summary of Connector Commands.
- *value* is the value that the field must have to result in an evaluation of true. If you want to allow any value for the field to evaluate as true, you can use the asterisk (*****) as a wildcard.

– An *action* instructs the simulator what action to take when the requisite comparison or comparisons evaluate as true.

  An *action* uses the following format:

  <action type="*type*" field="*field*" value="*value*" />

  where:

  - *type* is a predefined action that the command can take. For more information, including the types that are valid for each command, see Summary of Connector Commands.
  - *field* is either blank or the name of a known variable or variable field in Dialog Designer.
  - *value* is the value to be assigned to the variable or variable field designated in the field element.

  A special case action is the error condition. To simulate an error for any command, use the following general format:

  <action type="ERROR" field="" value="*errorMessage*"/>

  Note that the **field** element is empty. When the simulator encounters an action with this format, the simulator returns the *errorMessage*.

For more information about other configuration options, see Additional Configuration Options in Connector Scripts.

Additional Configuration Options in Connector Scripts

In addition to the basic configuration options available with *comparison* and *action* tag sets, you can use other options to refine the way your scripts work.

See the following sections:

Combining Comparisons and Actions

You can use multiple comparison and action tags to perform multiple comparisons or actions in the same configuration. Using multiple comparisons and actions is equivalent to performing a Boolean *and* Operation: For the actions to be performed, all comparisons must first evaluate as true. When all comparisons evaluate as true, then all actions are performed.

For example, consider the following snippet of code:

```
<configuration>
   <compare type="EQUAL" field="VDUID" value="1122334455"/>
   <compare type="EQUAL" field="workflow" value="myworkflow"/>
   <action type="SET_OUTPUT" field="workflowoutput"
     value="someValue"/>
   <action type="SET_VDU" field="testfield" value="anotherValue"/>
</configuration>
```

In this configuration, the script first checks the **id** field of the **vdu** variable, to see if it contains the value **1122334455**. If it does not, then the command does not execute. If it does, the script continues to the next comparison. The second comparison checks to see if the **workflow** variable has a value of **myworkflow**. If it does, the script then performs both actions.

In this example, the actions the script takes are to:

- Set the **workflowoutput** variable to the value defined by the first action **`value`** element (*someValue*).
- Set the **testfield** field of the vdu variable to the value defined by the second action **`value`** element (*anotherValue*).

Selecting One Comparison/Action Set Instead of Another

You can use multiple configurations to evaluate a series of comparisons and perform a particular action or set of actions. Using multiple configurations is equivalent to performing a Boolean OR operation: For the actions to be performed, one particular comparison or set of comparisons must evaluate as true.

Configurations are evaluated in order, so you must take care when placing each configuration to be evaluated. As soon as the simulator evaluates a configuration as true, the simulator performs the corresponding actions. The simulator ignores all remaining configurations for that command.

For example, consider the following snippet of code:

```
<command name="Workflow">
   <configuration>
      <compare type="EQUAL" field="VDUID" value="1122334455"/>
      <action type="SET_OUTPUT" field="workflowoutput"
        value="someValue"/>
   </configuration>
   <configuration>
      <compare type="EQUAL" field="VDUID" value="*"/>
      <action type="SET_OUTPUT" field="workflowoutput"
        value="anotherValue"/>
   </configuration>
</command>
```

In this example, the simulator evaluates the first configuration. If the value of the **vdu** variable **id** field is **1122334455**, the simulator sets the value of the **workflowoutput** field to the first value (*someValue*). The simulator then considers this command completed and the application continues.

If the value of the **vdu** variable **id** field is anything *other than* **1122334455**, the simulator continues to the second configuration. The second configuration uses the asterisk (*) wildcard for the **value** element. This value means that the second configuration accepts any value for the **vdu** variable **id** field. In this case, the simulator sets the value of the **workflowoutput** field to the second value (*anotherValue*).

Tip:

In this example, because any other value for the vdu variable id field is acceptable, you can also use the key word ALL. In this example, that means you can substitute the code **<compare type="ALL" />** in place of the line:

```
<compare type="EQUAL" field="VDUID" value="*"/>
```

You can use this substitute line of code as a "catch-all" for all those conditions that do not match with any specific value.

Sample Connector Scripts

This section contains two sample scripts that illustrate the guidelines established in Connector Script Guidelines. You can use these sample scripts as models to create your own scripts. The scripts include:

- Sample IC Connector Script
- Sample CTI Connector Script

Sample IC Connector Script

The following script is a sample of an IC connector script. This script tells the simulator to check whether a new call is coming in on channel 10. If the call is coming in on channel 10, the script instructs the simulator to set the **id** field of the **vdu** variable to the value **1122334455**. If the call is coming in on any other channel, the script instructs the simulator to set the **id** field of the **vdu** variable to the value **6677889900**.

```
<?xml version="1.0" encoding="UTF-8"?>
<simulator>
  <connector name="IC">
    <command name="NewCall">
      <configuration>
        <compare type="EQUAL" field="channel" value="10"/>
        <action type="SET_VDU_ID" field="" value="1122334455"/>
      </configuration>
      <configuration>
        <compare type="EQUAL" field="channel" value="*"/>
        <action type="SET_VDU_ID" field="" value="6677889900"/>
      </configuration>
    </command>
  </connector>
</simulator>
```

Sample CTI Connector Script

The following script is a sample of a CTI connector script. CTI connector scripts are used only to check for error responses when commands are invoked. If no error response is defined in the script for a particular CTI connector, the simulator uses the default action. The default action is usually the successful completion of the item.

The following script simulates various possible error responses when the simulator encounters the designated CTI commands.

```
<?xml version="1.0" encoding="UTF-8"?>
<simulator>
  <connector name="CTI">
    <command name="Hold">
      <configuration>
        <compare type="ALL"/>
        <action type="ERROR" field="" value="Error invoking hold"
/>
      </configuration>
    </command>
    <command name="Dial">
      <configuration>
        <compare type="EQUAL" field="PhoneNumber"
value="4085777734" />
        <action type="ERROR" field="" value="busy" />
      </configuration>
      <configuration>
        <compare type="EQUAL" field="PhoneNumber"
value="4085777735" />
        <action type="ERROR" field="" value="noanswer" />
      </configuration>
    </command>
    <command name="Retrieve">
      <configuration>
        <compare type="ALL" />
        <action type="ERROR" field="" value="Error invoking
retrieve" />
      </configuration>
```

```
    </command>
    <command name="Disconnect">
      <configuration>
        <compare type="ALL"/>
        <action type="ERROR" field="" value="Error invoking
disconnect"/>
      </configuration>
    </command>
    <command name="Conference">
      <configuration>
        <compare type="ALL"/>
        <action type="ERROR" field="" value="Error invoking
conference"/>
      </configuration>
    </command>
    <command name="Transfer">
      <configuration>
        <compare type="ALL"/>
        <action type="ERROR" field="" value="Error invoking
transfer"/>
      </configuration>
    </command>
  </connector>
</simulator>
```

This script performs the following actions in applications that use CTI connectors:

- If the command is Hold (CTI), the simulator returns an error message that says, “error invoking hold”.
- If the command is Dial (CTI), the simulator checks to see what number was dialed. If the number is 4085777734, the simulator returns an error of “busy”. If the number is 4085777735, the simulator returns an error of “noanswer”.
- If the command is Retrieve (CTI), the simulator returns an error message that says, “error invoking retrieve”.
- If the command is Disconnect (CTI), the simulator returns an error message that says, “error invoking disconnect”.

- If the command is Conference (CTI), the simulator returns an error message that says, “error invoking conference”.
- If the command is Transfer (CTI), the simulator returns an error message that says, “error invoking transfer”.

Summary of Connector Commands

This section lists and describes all the commands that can be used in conjunction with connector scripts. The commands are divided into the following groups:

- IC Connector Commands
- CTI Connector Commands

IC Connector Commands

The following table lists and describes the IC connector commands that can be included in connector scripts.

IC Connector Commands

Command Name	Valid Comparison Field Values	Valid Action Type Values	Default Action[1]
Alarm	alarmname - Contains the name of the alarm, as defined in the Alarm item	ERROR	The simulator returns an indication of a successful alarm.
CallGone	VDUID - Contains the value of the **vdu** variable **id** field	ERROR	The simulator returns a successful “call gone” response
GetVDU	VDUID - Contains the value of the **vdu** variable **id** field Name - Contains the name of the workflow	ERROR	The simulator looks up in the simulator VDU table the requested **id** value and returns the value. If the value is not in the VDU table, the simulator returns an error.
NewCall	Channel - Contains the value of the channel on which the call is coming in.	ERROR SET_VDU_ID	The simulator returns the value of the SET_VDU_ID action type to the value of the **vdu** variable **id** field.

IC Connector Commands (continued)

Command Name	Valid Comparison Field Values	Valid Action Type Values	Default Action[1]
SetVDU	VDUID - Contains the value of the **vdu** variable **id** field Name - Contains the name of the workflow	ERROR	The simulator sets the value of the **value** element in the simulator VDU table and returns a response to the effect that the value was set correctly.
Transfer	VDUID - Contains the value of the **vdu** variable **id** field	ERROR	The simulator returns a successful call transfer response.
InvokeWorkflow	VDUID - Contains the value of the **vdu** variable **id** field workflow - Contains the name of the workflow to invoke	ERROR SET_OUTPUT - The field for this type must be the name of a variable that matches the name of a variable on the IC system. SET_VDU - The field for this type must be the name of a custom field in the **vdu** variable.	The simulator returns a response to the effect that the workflow ran successfully. At the same time, the simulator sets the designated variables with the values from the SET_OUTPUT and SET_VDU action types.

1. The default action is the action that the simulator performs when no comparison evaluates as true.

For more information about the IC connector, see About the IC Connector.

CTI Connector Commands

The following table lists and describes the CTI connector commands you can include in connector scripts:

CTI Connector Commands

Command Name	Valid Comparison Field Values	Valid Action Type Values[1]	Default Action[2]
CallInfo	*	ERROR - The error code to be returned is defined in the *value* element.	The simulator returns a response that the call information was successfully obtained.
Conference	*	ERROR - The error code to be returned is defined in the *value* element.	The simulator returns a response that the call was successfully conferenced in.
Dial	PhoneNumber - Contains the value of the Dial Number variable from the Dial (CTI) item	ERROR - Error codes for return include: • busy • failed • noanswer • unreachable • unknown	The simulator returns a response that the number was successfully dialed (with an “established” state). Before transferring a call, the state of the Dialed call should be checked.
Disconnect	*	ERROR - The error code to be returned is defined in the *value* element.	The simulator returns a response that the call was successfully disconnected (with a “disconnected” state).
Hold	*	ERROR - The error code to be returned is defined in the *value* element.	The simulator returns a response that the call was successfully placed on hold (with a “held” state).

CTI Connector Commands (continued)

Command Name	Valid Comparison Field Values	Valid Action Type Values[1]	Default Action[2]
Retrieve	*	ERROR - The error code to be returned is defined in the *value* element.	The simulator returns a response that the call was successfully retrieved from hold status (with an “established” state).
Transfer	*	ERROR - The error code to be returned is defined in the *value* element.	The simulator returns a response that the call was successfully transferred (with a “transferred” state).

1. Use the value field to return the error code.

2. The default action is the action that the simulator performs when no comparison evaluates as true.

For more information about the CTI connectors, see About CTI Connectors.

Appendix F: Conditional Operators

Conditional operators are used within Avaya Dialog Designer applications with If items.

The following table lists and describes the function of each conditional operator. The table also provides information about how the **Condition** and **If** items respond to each operator.

Conditional Operators

Operator	Function/Comments	Response
Equal	Compares the value of two variable values, the "left" value and the "right" to see if they are equal. This operator assumes that the two values being compared are character or string values, and it compares the value of alphabetic characters taking case into account. In other words, this operator treats the letter “A” and the letter “a” as the different values.	• In the **Condition** item, if the two values are equal, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the two values are equal, Dialog Designer plays the associated prompt.
NotEqual	Compares the value of two variable values, the "left" value and the "right" to see if they are *not* equal. This operator assumes that the two values being compared are character, or string, values, and it compares the value of alphabetic characters taking case into account. In other words, this operator treats the letter "A" and the letter "a" as the different values.	• In the **Condition** item, if the two values are *not* equal, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the two values are *not* equal, Dialog Designer plays the associated prompt.

Conditional Operators (continued)

Operator	Function/Comments	Response
GreaterThan	Compares the value of two variable values, the "left" value and the "right" to see whether the value of the left variable is greater than the value of the right variable. This operator assumes that the two values being compared are character, or string, values, and it compares the value of alphabetic characters taking case into account. In other words, this operator treats the letter "A" and the letter "a" as the different values.	• In the **Condition** item, if the value of the left variable is greater than the value of the right variable, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the value of the left variable is greater than the value of the right variable, Dialog Designer plays the associated prompt.
LessThan	Compares the value of two variable values, the "left" value and the "right" to see whether the value of the left variable is less than the value of the right variable. This operator assumes that the two values being compared are character, or string, values, and it compares the value of alphabetic characters taking case into account. In other words, this operator treats the letter "A" and the letter "a" as the different values.	• In the **Condition** item, if the value of the left variable is less than the value of the right variable, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the value of the left variable is less than the value of the right variable, Dialog Designer plays the associated prompt.

Conditional Operators (continued)

Operator	Function/Comments	Response
EqualsIgnoreCase	Compares the value of two variable values, the "left" value and the "right" to see if they are equal. This operator assumes that the two values being compared are character, or string, values, and it compares the value of alphabetic characters without regard to case. In other words, this operator treats the letter "A" and the letter "a" as the same value.	• In the **Condition** item, if the two values are equal, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the two values are equal, Dialog Designer plays the associated prompt.
NotEqualsIgnoreCase	Compares the value of two variable values, the "left" value and the "right" to see if they are *not* equal. This operator assumes that the two values being compared are character, or string, values, and it compares the value of alphabetic characters without regard to case. In other words, this operator treats the letter "A" and the letter "a" as the same value.	• In the **Condition** item, if the two values are *not* equal, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the two values are *not* equal, Dialog Designer plays the associated prompt.
=	Compares the value of two variable values, the "left" value and the "right" to see if they are equal. This operator assumes that the two values being compared are number values.	• In the **Condition** item, if the two values are equal, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the two values are equal, Dialog Designer plays the associated prompt.

Conditional Operators (continued)

Operator	Function/Comments	Response
!=	Compares the value of two variable values, the "left" value and the "right" to see if they are *not* equal. This operator assumes that the two values being compared are number values.	• In the **Condition** item, if the two values are *not* equal, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the two values are *not* equal, Dialog Designer plays the associated prompt.
<	Compares the value of two variable values, the "left" value and the "right" to see whether the value of the left variable is less than the value of the right variable. This operator assumes that the two values being compared are number values.	• In the **Condition** item, if the value of the left variable is less than the value of the right variable, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the value of the left variable is less than the value of the right variable, Dialog Designer plays the associated prompt.
<=	Compares the value of two variable values, the "left" value and the "right" to see whether the value of the left variable is less than or equal to the value of the right variable. This operator assumes that the two values being compared are number values.	• In the **Condition** item, if the value of the left variable is less than or equal to the value of the right variable, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the value of the left variable is less than or equal to the value of the right variable, Dialog Designer plays the associated prompt.
>	Compares the value of two variable values, the "left" value and the "right" to see whether the value of the left variable is greater than the value of the right variable. This operator assumes that the two values being compared are number values.	• In the **Condition** item, if the value of the left variable is greater than the value of the right variable, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the value of the left variable is greater than the value of the right variable, Dialog Designer plays the associated prompt.

Conditional Operators (continued)

Operator	Function/Comments	Response
>=	Compares the value of two variable values, the "left" value and the "right" to see whether the value of the left variable is greater than or equal to the value of the right variable. This operator assumes that the two values being compared are number values.	• In the **Condition** item, if the value of the left variable is greater than or equal to the value of the right variable, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the value of the left variable is greater than or equal to the value of the right variable, Dialog Designer plays the associated prompt.
HasMore	Checks a collection associated with a variable to see if there are any more items in the collection that have not been processed.	• In the **Condition** item, if there are more items to be processed, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if there are more items to be processed, Dialog Designer plays the associated prompt.
HasNoMore	Checks a collection associated with a variable to see if there are any more items in the collection that have not been processed.	• In the **Condition** item, if there are no more items to be processed, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if there are more items to be processed, Dialog Designer plays the associated prompt.
IsEmpty	Checks a variable, the “left” variable, to see if it has any value assigned. If it does not have a value assigned, the operator returns a value of **true**.	• In the **Condition** item, if the variable has no value assigned, that is, it is an empty variable, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the variable is an empty variable, Dialog Designer plays the associated prompt.

Conditional Operators (continued)

Operator	Function/Comments	Response
IsNotEmpty	Checks a variable, the “left” variable, to see if it has any value assigned. If it does have a value assigned, the operator returns a value of **true**.	• In the **Condition** item, if the variable has a value assigned, that is, it is not an empty variable, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the variable is not an empty variable, Dialog Designer plays the associated prompt.
IsTrue	Checks a variable, the "left" variable, to see it if has a value of **true**. This operator assumes that the variable is a Boolean variable.	• In the **Condition** item, if the variable has a value of **true**, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the variable has a value of **true**, Dialog Designer plays the associated prompt.
IsFalse	Checks a variable, the "left" variable, to see it if has a value of **false**. This operator assumes that the variable is a Boolean variable.	• In the **Condition** item, if the variable has a value of **false**, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the variable has a value of **false**, Dialog Designer plays the associated prompt.
IsAfter	Checks to see if the “left” variable has a date/time value that is later than the “right” variable. This operator assumes that both variables are of the same date or time format.	• In the **Condition** item, if the left variable has a date/time value later than the right variable, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the left variable has a date/time value later than the right variable, Dialog Designer plays the associated prompt.
IsBefore	Checks to see if the “left” variable has a date/time value that is earlier than the “right” variable. This operator assumes that both variables are of the same date or time format.	• In the **Condition** item, if the left variable has a date/time value earlier than the right variable, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the left variable has a date/time value earlier than the right variable, Dialog Designer plays the associated prompt.

Conditional Operators (continued)

Operator	Function/Comments	Response
IsDigits	Checks to see if the variable is of type **Digits**. That is, it checks to see if all the characters of the variable are digits.	• In the **Condition** item, if the variable is of type **Digits**, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the variable is of type **Digits**, Dialog Designer plays the associated prompt.
isNull	Checks to see if the value of the variable is "__DD_NULL".	• In the **Condition** item, if the variable is of type **setNull**, the call flow proceeds to the node designated in the **Next Form** property. • In the **If** item, if the variable is of type **setNull**, Dialog Designer plays the associated prompt.

Appendix G: CTI and IC Connectors

Dialog Designer applications offer the capability to interface and integrate with two external types of systems: Avaya Interaction Center (IC) systems and computer telephony integration (CTI) systems.

This appendix provides detailed information about using the built-in connectors to integrate with these systems in Dialog Designer applications. It contains the following sections:

About CTI Connectors

Dialog Designer applications can integrate with computer telephony integration (CTI) servers to extend the capabilities of VoiceXML 2.1 speech applications. Among the limitations of the W3C VoiceXML 2.0 Recommendation is the fact that it has very limited call control functionality. Supported CTI servers are AES 3.1 or AvayaCT 1.3.

With CTI connectors in Dialog Designer applications, you can add call control functionality, such as:

- Placing a caller on hold
- Doing a controlled call transfer
- Conferencing a call

To use this extended call control functionality in Dialog Designer, however, you must first enable CTI and add at least one computer telephony server to handle the extended functionality. These actions are accomplished in the project properties dialog box.

For the procedures to enable CTI, see CTI Tab.

CTI Connectors in Dialog Designer

Dialog Designer provides several CTI connectors to use when you want to incorporate CTI functionality in your applications:

- **Conference Party** – Makes it possible to add an additional caller to an existing conference. This additional caller is added to the conference without announcement (unannounced) to the existing callers. Sometimes, this type of call operation is also called *single-step conference*. See Conference Party (CTI).
- **Blind Call** - Used to transfer the caller to a destination number without knowing the call results. If the transfer fails, the call is lost and the application cannot retrieving it. See Blind Call (CTI).
- **Bridged Call** - Used to transfer the caller to a destination number while maintaining control of the call. If the call is not successful, the caller will be pulled back from being on hold and the call results can be analyzed for further action. See Consultation Call (CTI).
- **Call Info** - Collects information about the call and stores the information in a variable.

  This is most useful when passing call data information from a CTI parent application to a CTI module. See Call Info (CTI).
- **Dial** - Dials a telephone number. See Dial (CTI).
- **Conference** - Takes two telephone numbers passed to it and merges them in a single call session. See Conference (CTI).
- **Transfer** - Transfers a call and associated data to another number. See Transfer (CTI).
- **Disconnect** - Disconnects a designated number from the call. See Disconnect (CTI).
- **Hold** - Places the caller on a designated number on hold. See Hold (CTI).
- **Retrieve** - Releases a caller on hold from hold status. See Retrieve (CTI).

Note that the Data Node is the only node in which you can use CTI connector items.

Using CTI Connectors in Applications

Before you can use CTI in your applications, you must enable the CTI connector functionality. See Enabling CTI Functionality in Dialog Designer.

One of the most common uses of CTI in speech applications is to perform call transfers in which variable data can be transferred along with the call itself. For a suggested procedure to construct a call and data transfer using CTI connectors, see Constructing a Call and Data Transfer Using CTI Connectors.

If you plan to deploy your application that uses CTI connectors to an Avaya IVR system, such as Avaya IR or Avaya Voice Portal, you must perform some procedures to enable the Dialog Designer application to interact correctly with the IVR system. See Configuring Dialog Designer CTI Applications to Work with Avaya IVR Systems.

Enabling CTI Functionality in Dialog Designer

To use CTI connector functionality in Dialog Designer applications, you must first enable the CTI connector servlet in the Tomcat server engine. This task is accomplished in the project properties dialog box. To enable the CTI connector servlet, see CTI Tab.

Once the CTI connector is installed in Tomcat, when Tomcat is restarted, the CTI connector attempts to establish a connection with the CTI simulator in Dialog Designer. This connection allows you to test and simulate CTI functions in your Dialog Designer applications.

When you enable CTI for an application, Dialog Designer automatically uses the CTI connector servlet to collect call info data and store it in a special variable named **cticallinfo**. See About the cticallinfo Variable.

When you deploy a completed speech application, the CTI connector servlet must be redirected so that it no longer attempts to establish a connection with the CTI simulator in Dialog Designer. Rather, it should establish a connection with the actual telephony server (T-server).

Constructing a Call and Data Transfer Using CTI Connectors

One of the most common uses of CTI in speech applications is to perform call transfers in which variable data can be transferred along with the call itself. This type of transfer is different from the VXML call transfer capabilities of Dialog Designer. The two standard call transfer types, Blind Transfer and Bridged Transfer, both transfer only the call. They do not provide the capability to transfer data with the call.

When constructing a call transfer using CTI connectors, there are several guidelines to keep in mind. The following example is not the only way to construct such a call, but is one suggested way to construct such a call. The following process provides the general steps and guidelines to keep in mind when setting up a transfer call with CTI connectors. The following process does *not* mention all the prompts, announcements, and other elements you may need to produce a complete and functional speech application.

Note:
Support for call transfer functions can vary from one IVR system to another. For more information and details about support for transfer functions on your system, see the documentation for your IVR system.

Note:
This procedure can be simplified using Blind Call (CTI) and Consultation Call (CTI).

1. Set up the call for transfer:

a. Place a Data Node in your application call flow.

   For the sake of our example, we will call this **XferSetup**.

b. To open the **XferSetup** subflow editor, double-click the node.

c. Place a Hold (CTI) item after the **Next** item.

   This item keeps the original caller on hold while the system dials the telephone number to transfer to.

   Set this item with the following properties:

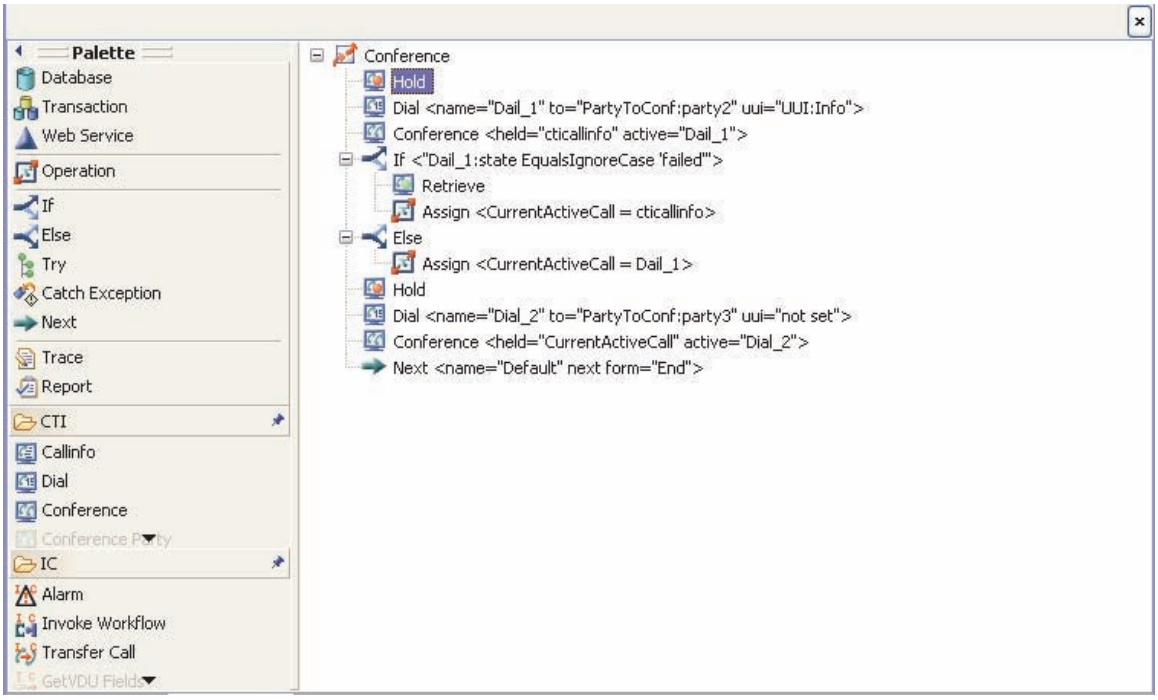
   - **Name** = **HoldInfo**
   - **Call ID Variable** = **cticallinfo**
   - **Call ID Variable Field** = (leave blank)

d. Place a Dial (CTI) item after the **Hold** item.

   This item uses the value of the **Dial Number Variable** and field to dial the destination telephone number. (The **Dial Number Variable Field** is used only if **Dial Number Variable** is a complex variable.)

e. To clean up below screenshot

CTI Conference Example



Note:

If you have collected data that you want to transfer with the call, make sure you assign it to a variable. Then, assign that variable to the **UUI Variable** property of the **Dial** item, in the **Avaya Properties** view.

Set this item with the following properties:

- **Name** = **DialCall**
- **Dial Number Variable** = The variable that contains the telephone number to be dialed. If the selected variable is a complex variable, you must also select a **Dial Number Variable Field**.
- **Dial Number Variable Field** = The variable field that contains the telephone number to be dialed. This field is available and required only if the variable selected for the **Dial Number Variable** is a complex variable.
- **UUI Variable** = The variable that contains any user-to-user (UUI) information you want to pass to the transfer destination.

  UUI is text-based data that accompanies the call. For example, during the course of a call, you might have collected information from the caller such as an account number, customer preferences, customer identification data, and so forth. You can use CTI transactions to pass this data to a call center agent, where this data can be used to populate a data window on the agent's computer screen.

  If the selected variable is a complex variable, you must also select a **UUI Variable Field**.
- **UUI Variable Field** = The variable field that contains the UUI information you want to pass to the transfer destination. This field is available and required only if the variable selected for the **UUI Variable** is a complex variable.
- **Ring max** = Enter the number of rings to allow on the far end before the system considers the call a "no answer" and returns the caller to this application for further handling.

f. Place a Condition item after the **Dial** item and set it up to direct the call flow to a node that will handle the call if the dialing action is completed successfully.

   For the sake of this example, we will call this condition **CallCompleted**. Set this condition item with the following properties:

   - **Operator** = **EqualsIgnoreCase**
   - **Left variable** = The variable associated with the **Dial** item (in this case, **DialCall**)
   - **Left variable field** = **state**
   - **Next Form** = The form to go to if the condition evaluates as 'true'

Note:
The “right” value, whether it is a variable or constant, is the condition that must be ’true’ before the call is passed to the **Next Form**. In this case, we are using the constant value so that, if the call is established, the application can pass control of the call on. If not, the default branch is used.

- **Right variable** = blank
- **Right variable field** = blank
- **Right Constant** = **established**

2. Accomplish the call transfer:
   a. Return to the Call Flow Editor and place another **Data** node in the main workspace.

      For the sake of our example, we will call this **accomplishXfer**.

   b. Connect the **CallCompleted** branch of the **XferSetup** node to the **accomplishXfer** node.
   c. To open the **accomplishXfer** subflow editor, double-click the node.
   d. Place a Transfer (CTI) item in the node subflow.
   e. For the **Held Call Info Variable**, use the variable associated with the **Hold** item, in this case, the **cticallinfo** variable in **XferSetup**.
   f. For the **Active Call Info Variable**, use the variable associated with the **DialCall** item, in this case, the **Dial Number Variable**.
   g. Use the **Next** item to direct the call flow to the return node or to another node that will clean up after the call and terminate the call session.

Note:
Once the call is successfully transferred, no further interaction with the caller is possible. In effect, this is a blind transfer, and the caller is not returned to this application when the transfer call is completed. As soon as the transfer is complete, this application proceeds to whatever node you designate in this step.

3. Set up the call flow to handle unsuccessful transfer attempts:
   a. Return to the Call Flow Editor and place another **Data** node in the main workspace.

      For the sake of our example, we will call this **Unhold**.

   b. Connect the **Default** branch of the **XferSetup** node to the **Unhold** node.
   c. To open the **Unhold** subflow editor, double-click the node.
   d. Place a Retrieve (CTI) item in the node subflow.

      The **Retrieve** item takes the original caller off hold and allows you to redirect the call based on whatever reason the call was not transferred successfully.

      For the **Call Info Variable** property, select the variable associated with the **Hold** item, in this case, the **cticallinfo** variable in **XferSetup**.

e. Return to the Call Flow Editor and place a Prompt and Collect Node in the main workspace.

   For the sake of our example, we will call this **GetWhatNext**.

f. Create and configure a prompt for the **GetWhatNext** node that tests for the possible return conditions of the **Dial** attempt, using Condition items. Each return condition can then have its own prompt to play to the caller.

   You can use the return condition also to redirect the call to a different node or whatever further action you want to take to complete the call.

About the cticallinfo Variable

When you enable CTI, Dialog Designer creates a complex variable, **cticallinfo**, which collects call data about the incoming call.

At runtime, when a call is initiated to the Dialog Designer application, the application automatically sends a **callinfo** command to the CTI connector at the beginning of the call. The T-server then returns the data to populate the fields of the **cticallinfo** variable.

The following table describes the fields associated with the **cticallinfo** variable.

cticallinfo Variable

Fields	Description
ani	Returns the ANI, or the number that the caller is calling from.
callid	Returns the call identifier created by the system when the call is established. The CTI connector uses this to keep track of call progress.
dnis	Returns the DNIS, or the number that the caller dialed.
state	Returns the current state of the call. Possible values include: • established • ringing • disconnected • hold • queued • failed • transferred
stationextension	This field is not currently used by Dialog Designer.
ucid	Returns the universal call identifier. This is a unique call ID recognized by all systems. You can use this to query other systems for information about the call.
uui	Returns the contents of the **uui** (user-to-user information) variable field.

Configuring Dialog Designer CTI Applications to Work with Avaya IVR Systems

To deploy a speech application that uses CTI connectors to an Avaya IVR system, either Avaya IR or Voice Portal, see Configuring CTI Settings.

Configuring Channel to Extension Mapping Script for Avaya IR Systems

To generate the channel to extension mapping for Avaya IR systems, see Configuring IR Channel Map.

About the IC Connector

Avaya Dialog Designer IVR applications can integrate with Avaya Interaction Center (IC) systems (such as IC 6.1.3-7.1), through the IC VOX server. This integration means that each system can leverage the strengths of the other system in a single speech application.

With the IC connector items in Dialog Designer IVR applications, you can:

- Raise alarms on the IC system. See Alarm.
- Transfer calls to the IC system. See Transfer Call.
- Invoke a workflow on the IC system. See Invoke Workflow.
- Get and set values in the vdu fields on the IC VOX server. See vdu and vdu_cache Variables

Note:
For the Avaya Interaction Center VOX Server to recognize the IVR system, it must be appropriately configured. The configuration parameters for the VOX Server are described in the *Avaya Interaction Center VOX Server Programmer's Guide*. Further, see Simulators Preferences for details on creating a VOX server simulation environment.

To use IC connector functionality in Dialog Designer applications, you must first enable the IC connector servlet in the Tomcat server engine. This task is accomplished in the project properties dialog box. To enable the IC connector, see IC Tab.

Once the IC connector is installed in Tomcat, when Tomcat is restarted, the IC connector attempts to establish a connection with the IC simulator in Dialog Designer. This connection means you can simulate and test IC functions in your Dialog Designer applications.

When you deploy the completed application, you must redirect the IC connector so that it no longer attempts to establish a connection with the IC simulator in Dialog Designer. Rather it must try to establish a connection with the actual VOX server on the IC system.

To redirect the IC connector to the VOX server:

Note:
The following procedure assumes that you have already copied and deployed the application to the application server. See Deploying the Application.

1. Locate the IC connector **web.xml** file.

   Following are the paths to the different application servers:

   - On IBM WebSphere and WebSphere Express servers (5.x), at:

     ***WebSphereHome*\AppServer\config\cells*cellname*\applications\CTIConnector_Application.ear\deployments\CTIConnector_Application\cticonnector.war\WEB-INF**

   - On IBM WebSphere and WebSphere Express servers (6.0 and above), at:

     ***WebSphereHome*\AppServer\profiles*servername*\config\cells*cellname*\applications\CTIConnector_Application.ear\deployments\CTIConnector_Application\cticonnector.war\WEB-INF**

   - On BEA WebLogic at:

     ***WebLogicHome**user project*\domains*domain*\servers*server*\stage\cticonnector\cticonnector.war\WEB-INF**

2. Use a text editor to open and edit the **web.xml** file, as follows:

   a. Locate the following parameters:

      <param-name>hostAddress</param-name>

      <param-value>localhost</param-value>

      <param-name>port</param-name>

      <param-value>3000</param-value>

      <param-name>defaultTimeout</param-name>

      <param-value>4000</param-value>

      <param-name>defaultTimeout</param-name>

      <param-value>4000</param-value>

      <param-name>trace.verbosity</param-name>

      <param-value>1</param-value>

   b. Replace **hostAddress** with the fully qualified domain name or IP address of the VOX server in the IC system.

   c. Replace **port** with the VOX listener port.

   d. Replace **defaultTimeout** (in milliseconds) with a configurable timeout that the IC Connector will wait on the VOX to respond after it was sent an IC command. Be very careful in changing this value as it could have an adverse affect on your system!

e. The **verbosity** ranges from 0 (off) -> 3 (full verbosity). 0 is recommended for production systems; 3 is recommended for troubleshooting problems.

3. Save your changes and close the **web.xml** file.
4. Verify that the VOX server has been configured to integrate with the IVR system.

   For information about configuring the VOX server, see the IC system documentation.
5. Restart the servlet container (Tomcat or WebSphere).

Once these changes have been made, the IC connector attempts to establish contact with the designated VOX server in the IC system whenever the servlet container (Tomcat or WebSphere) is started. If the connection cannot be made, the connector tries again after a set timeout. The IC connector, however, does not notify the IVR system that the connection cannot be made until the IVR application attempts to actually use the connector.

Thereafter, whenever you start or restart the servlet container, the IC connector attempts to restore the connection to the VOX server on the IC system. When the Dialog Designer application needs to send a command to the VOX server, the application locates and uses the IC connector.

vdu and vdu_cache Variables

When you enable IC, Dialog Designer creates two complex variables, **vdu** and **vdu_cache**, which map to an EDU (Electronic Data Unit) on the IC system.

The **vdu** variable is created with one read-only variable field, labeled **id**. At runtime, when a call is initiated to the Dialog Designer application, the application sends a **new call** command to the IC system. The IC system then returns a unique EDU ID value, in the form of a **vduid** variable value, which is stored in the **id** field of the **vdu** variable in the Dialog Designer application.

The **vdu_cache** variable mimics the **vdu** variable, but fields cannot be added, deleted, or renamed from in it. As changes are made to the **vdu** variable, the **vdu_cache** variable will also reflect these changes. The **vdu_cache** does not go to the VOX server to get the data *except* for the first time you access it.

You can add other fields to the **vdu** variable and use them to:

- Make variable values available to the IC system.

  When you create a custom field for the **vdu** variable, Dialog Designer makes the field's value available to the IC system through the IC connector. The IC system obtains the value of the variable field with the **VOX.setvdu** method. For more information see the *Avaya Interaction Center VOX Server Programmer's Guide*.
- Obtain variable values from the IC system.

  The IC system uses the **VOX.getvdu** method to pass variable values to the Dialog Designer application. For more information see the *Avaya Interaction Center VOX Server Programmer's Guide*.

Note:

To get values from or set values to the IC system, you must have variable fields in your **vdu** variable whose names *exactly* match the names of **vdu** fields on the IC system. There is no way to query the IC system for these field values, so you must be familiar enough with the workflows on the IC system to have the exact names of the fields whose values you want to use.

For more information about the use of the **vdu** or **vdu_variable** variable in Dialog Designer, see Invoke Workflow and Transfer Call.

For more information about variables in general, see Working with Variables.

IC Connector at Runtime

At runtime, the IVR system that uses the Dialog Designer speech application configured for IC and the IC system must both be configured to monitor the same channel for incoming calls. The IVR system can be either an Avaya IR or an Avaya Voice Portal system.

As a call comes in on a channel that the IC system is monitoring, the switch notifies the IC system. When the call is recognized on the IVR system, the Dialog Designer application running on the IVR system sends a **new call** command to the IC system. The IC system, in return, creates a new EDU and returns the ID to the application on the IVR system, where it populates the **vdu id** variable field value. This establishes a pipeline to the IC system for the call, thus making IC functionality available for use during that session.

Note:

The Avaya Voice Portal system does not use the concept of channels in routing calls, but rather of extensions. So, if you are using the IC connector in an application on a Voice Portal system, the channel that the IC system is monitoring on the switch must be mapped to a specific extension on the Voice Portal system. That is, if the IC system is using channel **1234**, the extension you use on the Voice Portal system must be **1234**. See the Avaya Voice Portal documentation library.

If this dual connection cannot be established within a period of time configurable on the IC system, the IVR system generates an error and discontinues the call.

At the end of the call, the speech application sends a **call gone** command to the IC system to signal that it can also disconnect the call.

IC will set the session.lasterror field when an error occurs during execution of an IC command. This field can be checked to obtain more detailed error information. If empty, no error has occurred.

You may also choose to disable runtime exceptions by setting the flag **sage.ic.throwexceptions** to false (default is true) in the speech application's web descriptor parameters. When an IC error has occurred, you can check the **lasterror** field to determine if the IC call was successful.

In Dialog Designer release 4.0, the preferred way to catch errors is to “catch” **SCERuntimeException**. See Catch (VXML Events).

WARNING:
If IC exceptions are disabled, you must check this field after every call or you risk missing errors that might have occurred.

Virtual Channel Mapping in IR

Two or more IR systems may send calls to an appserver. Problems in both the IC and CTI connectors will arise when the channels overlap and calls may be sent to your application on the same channel although map to a different extension. To get around this problem, you can create a virtual channel mapping that will allow you to map the physical channel to a logical one of a particular IR system. See Configuring IR Channel Map.

About IC Failover

Fail over is implemented by configuring the IC Connector so that the VOX server initiates the connection to the IC Connector. On the IC server, multiple VOX servers are set up so in case one goes down, another one can be started up. When the IC Connector starts up, it waits for a VOX server to initiate a connection before it is able to serve calls.

If the currently connected VOX server goes down, the IC Connector will detect that the VOX connection is inactive and then goes back and waits for a new VOX to initiate a connection before IC Connector can serve calls again. It is up to the user to make sure a new VOX server is started in case one goes down.

Glossary

ADR	*See* application detail record (ADR).
ANI	*See* automatic number identification (ANI).
API	*See* application program interface (API).
application detail record (ADR)	Data records which contain historical information about an application used as part of a session. These records include information such as the session ID number, a timestamp, and a “friendly name” string determined by the developer who created the application.
application program interface (API)	A set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks.
application server	A computer on which the Avaya Dialog Designer speech application resides and runs. This computer is also where the Dialog Designer run-time libraries are installed, thus making it possible to run Dialog Designer applications on that server. The IVR system must be configured to start the speech application from this location.
ASR	*See* automated speech recognition (ASR).
automatic number identification (ANI)	A service that provides the originating telephone number of a call coming in to the system.
automated speech recognition (ASR)	Technology that employs a computer to recognize spoken words and respond appropriately.
call flow	As implemented in speech applications, the call flow determines how a call is handled when it enters an interactive voice response system, based on options offered to callers and their responses to those options.
CCXML	Call Control eXtensible Markup Language. An emerging XML specification, being developed to work in conjunction with VoiceXML and which addresses some of the technical limitations of VoiceXML. It enables the processing of asynchronous events, filtering and routing of incoming calls, and placement of outbound calls. Note that it is not intended to replace VoiceXML but rather to supplement it. See Ian Moraes’s excellent article, “VoiceXML, CCXML, SALT: Architectural Tools for Enabling Speech Applications,” on the Internet.
Computer Telephony Integration (CTI)	Software technology that integrates the use of telephones and computers without the need for special telephones, connectors, computer circuit packs, or other specialized hardware.
CTI	*See* Computer Telephony Integration (CTI).

dialed number identification service (DNIS) A service that identifies for the receiving system what telephone number was dialed by the caller. In the Voice Portal system this is often used to direct the call to a particular speech application, which is identified with that dialed number.

DNIS *See* dialed number identification service (DNIS).

DTMF See dual tone multi-frequency (DTMF).

dual tone multi-frequency (DTMF) The system used by touchtone telephones, DTMF assigns a specific frequency (consisting of two separate tones) to each telephone key on the keypad, so that it can easily be identified by a microprocessor.

Eclipse A Java-based open-source extensible IDE (integrated development environment) that provides application developers a feature-rich interface to develop their applications. Avaya Dialog Designer is designed as a set of Eclipse plug-in modules that make it possible for application developers to design and build speech applications without having to write the code manually.

gateway A network point that acts as an entry point to another network. In the context of Avaya Dialog Designer and VoIP applications, a gateway is the entry point, often associated with one or more switches, to the interactive voice response (IVR) system application environment.

grammar In the context of Dialog Designer, a grammar is are speech elements used in conjunction with automated speech recognition (ASR) technology. Grammars are lists of possible responses that callers might make when they respond to prompts using spoken replies. Grammars define which words or phrases the ASR engine can recognize and respond to.

H.323 A hierarchical, IP-based telephony standard for connecting IP telephones and speech applications to switches.

IC *See* Interaction Center (IC).

IDE *See* integrated development environment (IDE).

integrated development environment (IDE) A software application that usually provides a GUI environment, a text and/or code editor, a compiler and/or interpreter, and a debugger. This environment means that application or web developers can develop, test, and build their applications or Web sites within a single integrated space.

Interaction Center (IC) A multichannel contact management platform that enables businesses to align real-time contact center operations with business objectives.

interactive voice response (IVR) system A system, such as Avaya Voice Portal or Avaya IR, in which callers interact with a self-service application to get information, conduct transactions, or help with problems.

IVR system *See* interactive voice response (IVR) system.

JDBC An application program interface (API) specification in which programs written in Java connect with and access data contained in database programs using Structured Query Language (SQL).

localization	The process of modifying an application to operate and be understood in a different language, or locale. This usually involves modifying any phrases, prompts, and grammars associated with an application.
notebook	(Also known as a tabbed or stacked notebook) In the Eclipse context, a notebook is a set of views “stacked” on top of one another as a space saving measure. The views in the notebook are accessible by clicking tabs arranged along the top of the notebook. See the Eclipse documentation.
Open Speech Dialog Module (OSDM)	Speech application modules produced by Nuance software products, similar to application modules created with Dialog Designer. OSDMs can be used in Dialog Designer applications. (Dialog Designer supports the following OSDM versions: Address OSDM 2.0.3, Core OSDM 2.0.4, and Name OSDM 2.0.1.)
OSDM	*See* Open Speech Dialog Module (OSDM).
palette	In the Avaya Dialog Designer Editor views, this is the pane to the left of the view, in which editor options are displayed and selected.
Real-time Transfer Protocol (RTP)	A protocol for transmitting “real-time” data, such as audio or video data, across the Internet. This protocol does not guarantee “real-time” delivery of such data, but it does provide mechanisms to support data “streaming.”
RTP	*See* Real-time Transfer Protocol (RTP).
SCE	*See* service creation environment (SCE).
service creation environment (SCE)	A set of software tools used to develop, test, and debug speech applications. Avaya Dialog Designer is an SCE.
servlet	A small program that runs on a server, often Java-based.
servlet engine	A program that coordinates the overall operation and integration of a number of servlets. In the context of Avaya Dialog Designer, the supported servlet engines are Apache Jakarta Tomcat and IBM WebSphere/WebSphere Express.
Session Initiation Protocol (SIP)	A signaling protocol for the Internet that makes it possible to set up conferencing, telephony, events notification, and instant messaging. Within a VoIP framework, it initiates call setup, routing, authentication, to endpoints within an IP domain.
SIP	*See* Session Initiation Protocol (SIP).
speech user interface (SUI)	Any software interface in which the user interacts with the system using speech commands and audio prompts.
speech recognition	*See* automated speech recognition (ASR).
speech synthesis	*See* text-to-speech (TTS).
SQL	*See* Structured Query Language (SQL).
SSL	Secure Sockets Layer. A protocol for transmitting private data securely over the Internet. By convention, URLs that use SSL require a connection using the HTTPS protocol, rather than just HTTP.

SSML Speech Synthesis Markup Language.

A W3C standard designed to provide an XML-based markup language for assisting with the generation of synthetic speech in Web and other applications. The essential role of the markup language is to provide authors of synthesizable content a standard way to control aspects of speech such as pronunciation, volume, pitch, rate, and so forth, across different synthesis-capable platforms.

stacked notebook *See* notebook.

Structured Query Language (SQL) A standard interactive and programming language for getting data to and from a database.

SUI *See* speech user interface (SUI).

tabbed notebook *See* notebook.

TDD *See* Telecommunications Display Device (TDD).

Telecommunications Display Device (TDD) Sometimes designated as a teletypewriter (TTY) device, a telephone equipped with a keyboard and display, used by hearing- or speech-impaired callers to send and receive typed messages.

telephone user interface (TUI) Any software interface in which the user interacts with the system using a telephone or similar device.

teletypewriter (TTY) device *See* Telecommunications Display Device (TDD).

text-to-speech (TTS) Technology by which information in text format is rendered as audio output using a speech synthesis engine to simulate human speech.

TTS *See* text-to-speech (TTS).

TTY *See* Telecommunications Display Device (TDD).

TUI *See* telephone user interface (TUI).

VoiceXML (Sometimes presented as VXML) Voice eXtensible Markup Language.

A specification which provides for a user to interact with Internet-based resources using voice recognition technology. Instead of a typical Web browser that requires a combination of HTML, keyboard, and mouse device, VoiceXML relies on an Internet voice browser and/or telephone. Using VoiceXML, the user interacts with the Web “page” by listening to audio outputs (either pre-recorded or using a technology such as TTS) and by submitting input in the form of the user’s natural speaking voice and/or manual responses, such as telephone key presses.

Web service A standardized way of offering Web-based applications or services. Since they are Web- and standards-based applications, delivered over the Internet, Web services make it possible for organizations to communicate and share data that use different file formats and programming languages.

workspace In Avaya Dialog Designer, the area within the Editor view used to build the functionality for the selected editor. For example, in the Call Flow Editor, this is the space to the right of the palette, in which you place the nodes that represent application functions.

WSDL Web Services Description Language.

An XML-formatted language used to describe a Web service's capabilities.

XML eXtensible Markup Language.

A specification for the presentation of Internet documents, one which expands on the capabilities of HTML. A pared down version of SGML (Standard Generalized Markup Language), XML makes it possible for designers to create their own customized tags, which in turn makes it possible to do things over the Internet that cannot be done using simple HTML.

Index

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X