

VoiceObjects 11

Object Reference

Object Reference

VoiceObjects 11.1

To ensure that you are using the documentation that corresponds to the VoiceObjects software you are licensed to use, compare this version number with the software version shown in the Help menu of the VoiceObjects software you are using.

Copyright

Copyright © 2001-2011 Voxeo Germany GmbH and its licensors. All rights reserved.

Published in Germany – Legal information 2011

Information in this document is subject to change without notice and does not represent a commitment on the part of Voxeo or any of its affiliated companies (collectively “Voxeo”). The software described in this document is furnished solely under a license agreement. The software may be used or copied only in accordance with the terms of the license agreement. You shall not reverse engineer the software or sub-license, distribute or make the software available to third parties by other means except as specifically allowed in the license agreement or by mandatory law. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose, without the express written permission of Voxeo.

Protected by German, European and US patents. Further patents pending.

Companies, names, and dates used in examples herein are fictitious unless otherwise noted. If such names affect copyrights or trademarks or others, please notify Voxeo by e-mail at support@voxeo.com.

Trademarks

VoiceObjects is a registered trademark. Any other trademarks, trade names or service marks mentioned in this document belong to their respective owners.

The material presented herein is based upon information that we consider reliable, but we do not represent that it is error-free and complete. Voxeo is not making any representation or granting any warranty with respect to such material, and the distribution of such material shall not subject Voxeo to any liability.

Third-Party Products

If Licensed Products are distributed together with third-party software or if this is contained in the Licensed Products, which are subject to additional license provisions, Licensee undertakes to observe the license provisions of the third party.

Explicit Copyright Notice

The VoiceObjects software includes software developed by the Apache Software Foundation (www.apache.org). Copyright © 1999-2011 – The Apache Software Foundation. All rights reserved.

Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S., other countries, or both.

Specific versions of the VoiceObjects software contain copyright material authorized by the Eclipse Foundation (www.eclipse.org), their contributors and others. All rights reserved.

Specific versions of the VoiceObjects software work with Microsoft Excel or make use of copyright material from Microsoft Corporation (www.microsoft.com). All rights to such copyright material rest with Microsoft. Microsoft and Excel are registered trademarks of Microsoft Corporation.

The VoiceObjects software is bundled with the software NuGram IDE Basic Edition developed by Nu Echo Inc. (www.nuecho.com). All title and copyrights in and to the software NuGram IDE Basic Edition, the accompanying printed materials, and any copies of the software are owned by Nu Echo Inc. and its suppliers.

Document Number: E-004-20111215-VO11

Table of Contents

TABLE OF CONTENTS **3**

ABOUT THE OBJECT REFERENCE **18**

Organization of this Guide 18
Typographical Conventions 18
Feedback and Questions 19

OBJECT TYPES OVERVIEW **20**

Object Categories 20

Components 21

OSDMs 22

Resources 24

Logic 25

Actions 26

Layers 26

Business Tasks 27

Additional Object Configuration 27

 **AUDIO** **28**

Overview 28
Dialog Flow Scenario 28
Object Definition 28

Audio Resource 29

Parameter Set 30

Alternative Texts 31

VoiceObjectsXML Definition 33

Audio 33

ParameterSet 34

Item 35

AltText 35

Object Interoperability 36
Object Naming Conventions 36

 **BUSINESS TASK** **38**

Overview 38
Dialog Flow Scenario 38
Object Definition 39

Declaration 40

Parameter Sets 40

Processing 41

Start business tasks 41

Copyright © 2001-2011 Voxeo Germany GmbH

Finish business tasks 41
Completion and status of a task 44
Handle more than one active task 46
Task data logging in Infostore 47
Business tasks and standard navigation 48
VoiceObjectsXML Definition 49
BusinessTask 50
ParameterSet 50
Item 51
Object Interoperability 51
Object Naming Conventions 51
CASE 52
Overview 52
Dialog Flow Scenario 52
Object Definition 52
Case Indicator 53
WHEN 54
ELSE 55
VoiceObjectsXML Definition 55
Case 55
CaseItem 56
Else 57
Object Interoperability 57
Object Naming Conventions 58
COLLECTION 59
Overview 59
Dialog Flow Scenario 60
Object Definition 60
Declaration 61
Formatting 62
VoiceObjectsXML Definition 63
Collection 63
Value 64
Working with Collections 64
Set and extract values 64
Find values 66
Manipulate rows and columns 66
Advanced collection operations 66
Usage Examples 66
Object Interoperability 67

Copyright © 2001-2011 Voxeo Germany GmbH

Object Naming Conventions 67

CONFIRMATION 69

Overview 69
Dialog Flow Scenario 69
Object Definition 70
Confirmation Request 71
Confirmation Grammar 73
Correction Request 74
Correction Mapping 75
Options 79
Usage Examples 80
Confirm a single input 80
Confirm multiple inputs 81
VoiceObjectsXML Definition 81
Confirmation 81
CorrectionItem 84
Object Interoperability 85
Object Naming Conventions 85

CONNECTOR 87

Overview 87
Dialog Flow Scenario 87
Object Definition 88
HTTP Connector Definition 88
Connector 89
Authentication 93
Process Notification 94
Wait Loop Audio 95
Java Connector Definition 96
Connector 96
Web Service Connector 101
Connector 102
Processing the Connector Object 110
Exchanging Data via an XML API 111
Request XML 111
Result XML 115
Error Handling 116
Java connectors 116
HTTP connectors 116
HTTP Connector Framework 117
VoiceObjectsXML Definition 120

Copyright © 2001-2011 Voxeo Germany GmbH

Connector 121
ConnectorItem 122
ParameterSet 124
Item 124
ResultMapping 124
ResultMappingItem 125
Object Interoperability 125
Object Naming Conventions 125

 **DATABASE 127**

Overview 127
Dialog Flow Scenario 127
Object Definition 128
Database Connection 129
Database 130
Process Notification 133
Wait Loop Audio 133
Processing the Database Object 134
Using Application Server Database Pooling 134
VoiceObjectsXML Definition 135
Database 135
DatabaseItem 136
ResultMapping 137
ResultMappingItem 137
Object Interoperability 137
Object Naming Conventions 138

EVENT HANDLING 139

Overview 139
Event Levels 139
Inheritance 140
Event Handling Definition 140
VoiceObjectsXML Definition 146
Server-Internal vs. Platform Events 148

 **EXIT 151**

Overview 151
Dialog Flow Scenario 151
Object Definition 153
Goodbye Message 154
Processing 155
Parameter Set 156
Options 157
VoiceObjectsXML Definition 157

Copyright © 2001-2011 Voxeo Germany GmbH

Exit ... 158
ParameterSet ... 158
Item ... 159
Object Interoperability ... 160
Object Naming Conventions ... 160
EXPRESSION ... 161
Overview ... 161
Dialog Flow Scenario ... 161
Object Definition ... 161
Declaration ... 162
Formatting ... 164
Usage Examples ... 165
VoiceObjectsXML Definition ... 167
Expression ... 167
Argument ... 168
Value ... 168
Object Interoperability ... 169
Object Naming Conventions ... 169
Function Library ... 170
FORMAT ... 203
Overview ... 203
Dialog Flow Scenario ... 203
Object Definition ... 204
Format ... 205
Text-To-Speech Formatting Types ... 207
Formatting options ... 210
Text-To-Audio Formatting Types ... 211
TTA – Literals/Digits ... 212
TTA – Literals/Digits (ASCII) ... 214
TTA – Words/Numbers ... 215
TTA – Complete Value ... 216
TTA – Files ... 217
TTA – XML ... 218
VoiceObjectsXML Definition ... 223
Format ... 223
FormatItem ... 224
Object Interoperability ... 225
Object Naming Conventions ... 226
GOTO ... 227
Overview ... 227

Copyright © 2001-2011 Voxeo Germany GmbH

Object Definition 227
Destination 227
Reference Library 228
VoiceObjectsXML Definition 229
Goto 229
Object Interoperability 229
Object Naming Conventions 230
GRAMMAR 231
Overview 231
Object Definition 231
Grammar 232
Voice/Text 233
DTMF 236
Preprocessing 236
Text-to-Grammar 238
Utterance Validation 245
Grammar Formats 246
VoiceObjectsXML Definition 248
Grammar 248
GrammarItem 249
GrammarDefinition 249
Object Interoperability 251
Object Naming Conventions 251
HYPERLINK 252
Overview 252
Dialog Flow Scenario 252
Object Definition 252
Destination 254
Activation Grammar 256
Presentation Output 257
Confirmation Request 258
Confirmation Grammar 259
Message 260
Options 261
VoiceObjectsXML Definition 262
Hyperlink 262
Object Interoperability 266
Object Naming Conventions 266
IF 267
Overview 267

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Flow Scenario 267
Object Definition 268
IF 269
THEN 269
ELSE 269
VoiceObjectsXML Definition 269
If 270
IfItem 271
Else 271
Object Interoperability 272
Object Naming Conventions 272
 **INPUT 273**
Overview 273
Dialog Flow Scenario 273
Object Definition 274
Input Request 275
Grammar 275
Result Handling 278
Options 283
VoiceObjectsXML Definition 285
Input 285
ResultHandling 287
Item 287
Object Interoperability 288
Object Naming Conventions 288
 **LAYER 290**
Overview 290
Dialog Flow Scenario 291
Object Definition 292
Define manual layers 293
Define automatic layers 293
State Indicator 293
States 294
Default State 295
Formatting 296
Options 297
Usage Examples 297
Manual layer Persona 297
Automatic Layer TimeOfDay 298
VoiceObjectsXML Definition 299

Copyright © 2001-2011 Voxeo Germany GmbH

Layer ... 299
LayerState ... 300
DefaultLayerState ... 301
Object Interoperability ... 301
Object Naming Conventions ... 301
LIST ... 303
Overview ... 303
Dialog Flow Scenario ... 303
Object Definition ... 305
List Messages ... 306
Navigation Grammar ... 306
Navigation Outputs ... 310
Data Definition ... 311
Content Formatting ... 313
Selection Commands ... 317
Options ... 319
VoiceObjectsXML Definition ... 321
List ... 321
ContentFormatting ... 325
ColumnGroup ... 325
Column ... 326
NavigationMessages ... 327
NavigationMessage ... 327
SelectionCommands ... 328
SelectionCommand ... 328
Usage Examples ... 329
Object Interoperability ... 329
Object Naming Conventions ... 330
LOG ... 331
Overview ... 331
Dialog Flow Scenario ... 331
Object Definition ... 331
Log ... 332
Usage Examples ... 335
Log to a file ... 335
Log to the media platform ... 336
Log to the OSDM log ... 338
Log to a custom database ... 339
Log to the system database ... 340
VoiceObjectsXML Definition ... 341

Copyright © 2001-2011 Voxeo Germany GmbH

Log 341
LogItem 342
Object Interoperability 343
Object Naming Conventions 343
LOOP 344
Overview 344
Dialog Flow Scenario 344
Object Definition 344
Condition Declaration 345
Object Sequence 346
Options 346
VoiceObjectsXML Definition 346
Loop 346
Object Interoperability 347
Object Naming Conventions 348
MENU 349
Overview 349
Dialog Flow Scenario 349
Object Definition 349
Welcome Message 350
Menu Items 351
Selection Request 355
Options 356
VoiceObjectsXML Definition 358
Menu 358
MenuItem 360
Object Interoperability 361
Object Naming Conventions 361
MODULE 362
Overview 362
Dialog Flow Scenario 362
Object Definition 363
Title 363
Welcome Message 364
Object Sequence 365
Goodbye Message 365
Options 366
Dialog end processing 366
Reference library 368
OSDM 368

Copyright © 2001-2011 Voxeo Germany GmbH

VoiceObjectsXML Definition 369
Module 369
Object Interoperability 370
Object Naming Conventions 370
NAVIGATION 372
Overview 372
Standard Navigation 372
Custom Navigation 379
VoiceObjectsXML Definition 382
Standard navigation 382
Custom navigation 382
 **OSDM 385**
Overview 385
Dialog Flow Scenario 387
Object Definition 388
Parameterization 389
Result Handling 394
Options 395
Additional Object Configuration 396
VoiceObjectsXML Definition 399
OSDM 399
OSDMItems 400
OSDMItem 401
OSDMParameter 402
Object Interoperability 402
Object Naming Conventions 403
 **OUTPUT 404**
Overview 404
Dialog Flow Scenario 404
Object Definition 404
Output 405
Random Prompting 408
SSML 408
HTML 410
Usage Examples 412
Random prompting 412
Custom layers 414
Lively welcome outputs 415
Occurrence level 415
Language layer 416
Output objects within other Output objects 417

Copyright © 2001-2011 Voxeo Germany GmbH

Advanced usage example: Local barge-in 418
VoiceObjectsXML Definition 419
Output 419
OutputItem 420
Object Interoperability 421
Object Naming Conventions 422
PAUSE 423
Overview 423
Dialog Flow Scenario 423
Object Definition 424
Welcome Message 425
Looped Pause Message 425
Final Pause Message 426
Wake Up Grammar 427
Return Message 428
Options 429
VoiceObjectsXML Definition 430
Pause 431
Object Interoperability 432
Object Naming Conventions 433
PLUG-IN 434
Overview 434
Dialog Flow Scenario 434
Object Definition 434
VoiceXML Resource 435
Parameter Set 437
Options 438
VoiceObjectsXML Definition 440
Plug-In 440
Value 441
ParameterSet 441
Item 442
Object Interoperability 442
Object Naming Conventions 443
PRE-/POSTPROCESSING 444
Overview 444
Defining a Precondition 444
Defining Pre-/Postprocessing 445
VoiceObjectsXML Definition 446
Pre-/postprocessing 446

Copyright © 2001-2011 Voxeo Germany GmbH

Preconditions 446
PROPERTIES 447
Overview 447
Properties 448
 **RECORDING 451**
Overview 451
Dialog Flow Scenario 451
Object Definition 453
Recording Request 454
Processing 454
Parameter Set 458
Meta Information on Recording Result 459
Usage Examples 459
VoiceObjectsXML Definition 459
Recording 460
Object Interoperability 461
Object Naming Conventions 461
 **RESOURCE LOCATOR 463**
Overview 463
Object Definition 463
Declaration 464
Accessing Hosted Resources 465
VoiceObjectsXML Definition 465
Resource Locator 466
Object Interoperability 466
Object Naming Conventions 467
 **SCRIPT 468**
Overview 468
Dialog Flow Scenario 469
Object Definition 470
Script Resource 471
Parameter Set 473
Process Notification 490
Wait Loop Audio 491
Formatting 492
Usage Examples 493
VoiceObjectsXML Definition 493
Script 493
Value 494
ParameterSet 495

Copyright © 2001-2011 Voxeo Germany GmbH

Item 495
Object Interoperability 496
Object Naming Conventions 496
SEQUENCE 498
Overview 498
Dialog Flow Scenario 498
Object Definition 498
Object Sequence 499
VoiceObjectsXML Definition 499
Sequence 499
Object Interoperability 499
Object Naming Conventions 500
SILENCE 501
Overview 501
Dialog Flow Scenario 501
Object Definition 501
Silence Duration 502
Advanced Configuration 502
VoiceObjectsXML Definition 502
Silence 503
Object Interoperability 503
Object Naming Conventions 503
TEXT 504
Overview 504
Dialog Flow Scenario 504
Object Definition 504
Text Resource 505
Parameter Set 507
Alternative Texts 507
VoiceObjectsXML Definition 509
Text 510
ParameterSet 511
Item 511
AltText 511
Object Interoperability 512
Object Naming Conventions 513
TRANSFER 514
Overview 514
Bridge transfer 514
Blind transfer 515

Copyright © 2001-2011 Voxeo Germany GmbH

Consultation transfer 515
CCXML transfer 516
Call transfer fundamentals 516
Handle bridge transfer events 517
Handle consultation transfer events 518
Dialog Flow Scenario 518
Object Definition 520
Transfer Notification 521
Processing 522
Parameter Set 525
Termination Grammar 526
Connect Audio 527
VoiceObjectsXML Definition 527
Transfer 527
ParameterSet 529
Item 529
Object Interoperability 530
Object Naming Conventions 530
TUNING 531
Overview 531
Tuning Definition 531
Options 532
VoiceObjectsXML Definition 533
Tuning Parameters 533
VARIABLE 544
Overview 544
Dialog Flow Scenario 544
Object Definition 545
Declaration 546
Formatting 546
VoiceObjectsXML Definition 548
Variable 548
Value 548
Object Interoperability 549
Object Naming Conventions 549
VIDEO 551
Overview 551
Dialog Flow Scenario 551
Object Definition 552
Video Resource 553

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter Set 554
Alternative Texts 555
VoiceObjectsXML Definition 557
Video 557
ParameterSet 558
Item 558
AltText 559
Object Interoperability 560
Object Naming Conventions 560
A - LANGUAGE CODES 561

Copyright © 2001-2011 Voxeo Germany GmbH

About the Object Reference

This *Object Reference* contains a detailed description of each object type available in VoiceObjects. It includes an overview of the purpose of each object type, dialog flow scenarios, usage examples, and a detailed description of how to define its relevant properties and settings within VoiceObjects Desktop and using VoiceObjectsXML.

Organization of this Guide

The *Object Reference* starts with an overview of all available dialog objects grouped into categories, followed by the detailed description of each object type – sorted in alphabetical order.

Each object description provides the following information as required:

Overview – A high-level overview about the purpose of the individual object.

Dialog Flow Scenario – One or more dialog flow scenarios that demonstrate the dialog interaction between the object and the caller.

Object Definition – A detailed description of how to define all relevant properties and settings of an object with VoiceObjects Desktop.

Usage Examples – Various usage examples how each object can be used in combination with other objects.

VoiceObjectsXML Definition – A description of how to define all relevant properties and settings of an object using VoiceObjectsXML.

Object Interoperability – A list of all types of objects that can reference the described object.

Object Naming Conventions – Guidelines regarding the naming of this object and the use of the description field provided for integrated documentation.

The *Object Refererence* covers the description of all *dialog* object types, as opposed to the configuration objects (projects, users, servers and services), which are described in detail in the *Desktop for Eclipse Guide*, the *Administration Guide* and the *Deployment Guide* respectively.

Appendix A – Language Codes contains a list of all language codes available in VoiceObjects together with the respective language they represent.

Typographical Conventions

This document contains the following typographical conventions:

Italic Font	Used to indicate names of applications, projects, objects, variables, files, and folders; output text, and book titles.
Bold Font	Used to indicate any screen terminology like names of windows, worksheets, editors, sections, boxes, tabs, fields, and menus.
`Courier New`	Used for grammar code.

All path specifications in this document use slashes (/) to apply to both Linux and Windows. If you work on Windows you may also use backslashes (\).

Copyright © 2001-2011 Voxeo Germany GmbH

Feedback and Questions

If you have any comments on this document please send your feedback to support@voxeo.com.

If you have technical difficulties please contact your local VoiceObjects administrator or if you have a valid software support and maintenance contract in place send an email describing your problem to support@voxeo.com.

Copyright © 2001-2011 Voxeo Germany GmbH

Object Types Overview

This overview provides a list of all dialog object types available in VoiceObjects and described in the *Object Reference*, and introduces additional object configurations such as event handling, navigation, and tuning properties.

Object Categories

All object types available in VoiceObjects are grouped into the following functional categories:

- **Components** – provide higher-level dialog objects
- **OSDMs** – provide pre-packaged dialog functionality
- **Resources** – provide access to external resources
- **Logic** – provide standard programming constructs to implement business logic
- **Actions** – provide interactive dialog control
- **Layers** – provide custom layers like persona or language
- **Business Tasks** – provide a business view on top of the dialog definition

Copyright © 2001-2011 Voxeo Germany GmbH

Components

Objects grouped into the Components category are higher-level objects, which provide dialog functionality frequently used within applications, such as menus, reading out lists and tables, summarizing and confirming multiple caller input etc. This category also offers the most basic objects necessary to interact with the caller, like the Output and Input object.

The objects of the Components category have various integrated system layers that provide for example sophisticated event handling and dialog navigation capabilities.

Icon	Object Name	Short Description
	**Module**	Represents the start node in a dialog flow and provides global settings, which are inherited by objects contained in the module. Typical global settings made here affect e.g. event handling, standard navigation, and ASR tuning.
	**Input**	Plays an Output to request information from the caller and assigns the caller's input to one or more Variable or Layer objects. The input can be either DTMF or recognized voice input in the voice and video channel, or text in the text and Web channel.
	**Output**	Presents any type of output to the caller. In the voice and video channel, outputs may contain text that is read out through a text-to-speech engine, or prerecorded audio or video files, possibly containing various dynamic content as well as Silence objects. In the text and Web channel, the output defines what is displayed on the screen of the mobile device.
	**Sequence**	Represents a dialog component to group multiple objects into a processing sequence.
	**Menu**	Presents a list of choices and lets the caller select one. When the caller has selected an item, the dialog continues with the processing of the object referenced in that item.
	**Confirmation**	Represents a configurable dialog component to summarize information collected in one or more previous dialog steps, and to let the caller confirm the summary or correct individual items (e.g. by using Input objects).
	**List**	Generates a dynamic and interactive dialog that presents the specified List items and provides list navigation and selection options to the caller. Supports both one-dimensional lists and two-dimensional tables, offering vertical and horizontal navigation, if required.

Copyright © 2001-2011 Voxeo Germany GmbH

OSDMs

The OSDMs category contains OpenSpeech™ DialogModules™ (OSDMs). OSDMs are building blocks for voice applications, which accelerate the development process by packaging application functionality for applications based on VoiceXML. They provide all prompts, grammars, dialog flow, and other parameters required to address frequent speech application tasks, like obtaining alphanumeric strings, dates, phone numbers, etc.

The following table lists all thirteen OSDM types available within the VoiceObjects platform together with short descriptions and alias names by which the types are referred to in the OSDM handbook.

Icon	Type	Alias	Description
	*Alphanumeric*	*Alphanum*	Collects a string of connected numbers and letters (i.e. the caller is not required to pause after each digit or letter). Since distinguishing individual letters is especially difficult, it is usually necessary to constrain the task by specifying a list of valid alphanumeric strings (e.g. 1000 possible account IDs).
	*Credit Card Expiration*	*CCExpDate*	Collects the expiration date for a credit card, usually a month and a year. The OSDM recognizes the last day of a given month as well.
	*Credit Card Number*	*CreditCard*	Collects a credit card number. Can understand card numbers for Visa, MasterCard, American Express, Discover, Diners Club, and many private label cards provided by retailers.
	*Currency*	*Currency*	Collects currency amounts in dollars and cents. You can constrain the range of currency amounts, specify the granularity, and specify a disambiguation mode.
	*Custom Context*	*CustomContext*	Allows you to create your own vocabulary and grammar to implement new functions not provided by the other OSDMs.
	*Date*	*Date*	Collects dates in several formats (e.g. “*June fourth*” or “*6/4/70*”). You can specify an entry format and constrain the range of dates accepted by the DialogModule.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Type	Alias	Description
	*Digits*	*Digits*	Fundamentally identical to the Alphanum DialogModule, except that it collects a string of connected digits only. To constrain the recognition task, you can specify a minimum and maximum number of digits to listen for (e.g. at least seven digits, but not more than ten). In most cases, this DialogModule is more accurate than Alphanum for digits-only collections.
	*Number*	*NaturalNumbers*	Collects any numerical amount. Callers can say "*fifty four hundred*" or "*fifty point twenty five*" instead of digits only.
	*Phone Number*	*Phone*	Collects a phone number in the North American Numbering Plan consisting of ten digits (long distance) with optional "1," seven digits (local) or three digits (411, 911, etc.) as configured. Phone numbers must be spoken digit-by-digit; one-eighthundred is supported as a special case. Certain illegal strings for area codes and exchanges are not allowed (e.g., 000-000-xxxx).
	*Postal Code*	*ZipCode*	Recognizes a five-digit or nine-digit US ZIP Code.
	*Social Security Number*	*SocialSecurity*	Collects a 9-digit US social security number. Illegal numbers, such as those beginning with three zeroes, are screened out.
	*Time*	*Time*	Collects a time of day. Callers can use 12-hour or 24-hour times, as well as prefix words such as "*about*" and "*around*" To constrain the recognition task, you can specify a range of valid times, granularity, and a disambiguation mode for determining whether to listen for times in the past or in the future.
	*Yes/No*	*YesNo*	Collects either an affirmative or negative response from the caller.

Copyright © 2001-2011 Voxeo Germany GmbH

Resources

Objects grouped into the Resources category provide access to external resources such as grammars, prerecorded audios, or existing VoiceXML files. It also covers objects for back-end access and custom logging.

Icon	Object Name	Short Description
	**Audio**	Provides access to prerecorded audio files. Audio files can cover spoken prompts, music, or sound effects. Not supported in the text and Web channel.
	**Video**	Provides access to video files. Used in multimedia services for 3G mobile devices. Not supported in the text and Web channel.
	**Text**	Provides access to text files.
	**Format**	Provides the concept of output formatting for dynamic content contained in Variable, Expression, Script, Collection, and Layer objects, which are presented to the caller in an Output object.
	**Silence**	Allows applications to play silence for a specified duration. Not supported in the text and Web channel.
	**Grammar**	Defines a recognition grammar, which can be used in all objects that accept caller input (such as Input, Menu, or Confirmation objects). A recognition grammar defines the input a caller can respond with to a particular request prompt (voice, DTMF, or text). Allows the definition of embedded grammars, references to external grammar files or grammars that are dynamically generated at call time.
	**Connector**	Allows access to external data sources and back-end systems to connect the dialog to dynamic content by providing an open Java, Web services and XML API for bi-directional data exchange. It can also be used to establish a connection to various other communication channels such as SMS, MMS, e-mail, fax or Web sites.
	**Database**	Allows access to relational database management systems to retrieve, store, or manipulate data.
	**Script**	Allows the specification of server-side scripting language code (JavaScript) to control any kind of complex business logic within the dialog flow.
	**Plug-In**	Provides access to an existing VoiceXML resource. Alternatively it can be used to plug-in custom VoiceXML code to take advantage of proprietary VoiceXML extensions of the underlying media platform. Not supported in the text and Web channel.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Object Name	Short Description
	**Log**	Provides the capability to write log messages to a file, to a custom database, to the media platform, or to the system database.
	**Resource Locator**	Represents a general access path to any kind of external resource in the same way that a URL (Uniform Resource Locator) does in the Web environment.

Logic

Objects grouped into the Logic category provide low-level elements based on standard programming constructs to control the logic of a dialog flow and the corresponding dialog context.

Icon	Object Name	Short Description
	**Variable**	Provides the concept of a variable as in any standard programming or scripting language to control the dialog context.
	**Collection**	Represents a collection of data values that can be accessed element by element with an index key similar to a database table or the construct of a hash table in any standard programming language.
	**Expression**	Provides a comprehensive set of expression functions, which can be used to perform, for example, arithmetical calculations, string operations, comparison operations, or any kind of Boolean operation.
	**If**	Allows you to specify a standard IF-THEN-ELSE construct as in any standard programming or scripting language.
	**Case**	Represents a standard CASE or SWITCH construct as in any standard programming or scripting language.
	**Loop**	Allows you to implement standard loop constructs of the type WHILE (condition) {…} and DO {…} WHILE (condition).
	**Goto**	Allows you to incorporate an explicit GOTO or GOSUB instruction within a dialog flow.

Copyright © 2001-2011 Voxeo Germany GmbH

Actions

Objects grouped into the Actions category are dialog elements that offer interactive dialog control to the caller, e.g. pausing an application, being transferred to a call center agent, or branching to a different part of the application.

Icon	Object Name	Short Description
	**Hyperlink**	Allows the caller to use a predefined command to branch out to a certain dialog component or subdialog at any point in time.
	**Pause**	Allows the caller to pause a call at any point in time with a predefined hyperlink command. It can also be used to pause the application at certain points in the dialog flow by design, e.g. to give the caller time to perform a task in a troubleshooting application before moving on to the next step. Not supported in the text and Web channel.
	**Recording**	Provides a mechanism to record a message from the caller, which can also be played back to the caller for review. Not supported in the text and Web channel.
	**Transfer**	Connects the caller to another entity (e.g. telephone line or another application). Not supported in the text and Web channel.
	**Exit**	Provides the termination of the connection between the phone of the caller and the application hosted on VoiceObjects Server.

Layers

The Layers category contains objects that provide custom layer definition capabilities.

Icon	Object Name	Short Description
	**Layer**	Lets the developer define typical custom layers like persona, customer/caller state, or time of day. These layers can be used throughout the application to allow for best practices of Voice User Interface design, like time-dependent greeting, persona-based prompts, menus depending on caller status etc.

Copyright © 2001-2011 Voxeo Germany GmbH

Business Tasks

The Business Tasks category contains objects that define a business view of an application on top of the dialog definition itself.

Icon	Object Name	Short Description
	**Business Task**	Represents a task the caller can accomplish while interacting with an application. Task-related data, i.e. application-specific information gathered while the task was processed and general statistics, such as whether a task was successfully completed, the time that a task was active, recognition rates, etc., will automatically be logged to Infostore.

Additional Object Configuration

Depending on the object type, the following additional sections are available in the object editors and also described in the *Object Reference*:

- **Precondition** – to associate a precondition with an object that activates or deactivates its processing dynamically at call time.
- **Pre-/Postprocessing** – to link additional objects to be processed before and/or after the processing of a specific object.
- **Event Handling** – to specify the handling of certain events that can occur during call time, like *No Match*, *No Input*, *Error - Media Platform*, etc.
- **Navigation** – to define standard and custom navigation commands available to the caller.
- **Tuning** – to set tuning properties that affect the behavior of the media platform with regard to speech recognition, speech synthesis etc.
- **Properties** – to set standard properties for an object, mainly for documentation purposes.

Copyright © 2001-2011 Voxeo Germany GmbH

Audio

Overview

The Audio object refers to prerecorded audio that can be played back through an Output object in the voice and video channel (the latter only if the video platform also supports playback of audio). It is possible to refer to a single file only, or to a set of files that hold variations of a prompt (for more information on randomizing, see Output). In addition, an audio generation engine can be referred to through an Audio object, in order to embed dynamically rendered audios into an application.
Alternative text can be provided, which is played back by text-to-speech (TTS) if the audio files are not available or not in the correct format.
The supported audio file formats depend on the underlying media platform that will do the final audio processing.

Using the Audio object, any kind of audio content can be played back in an application, such as speech, music, or earcons. For more details on audio design, see Creating Output in Appendix A – *Voice User Interface Design* in the *Design Guide*.

The Audio object, including the alternative text definitions, will be ignored in the text and Web channel.

The Object Definition below covers the configuration of the Audio object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

For information on how to manage the prompts of your application, refer to the *Storyboard Manager Guide*.

The Audio object belongs to the object category Resources.

Dialog Flow Scenario

The following dialog flow example demonstrates an Output object with two embedded Audio objects.

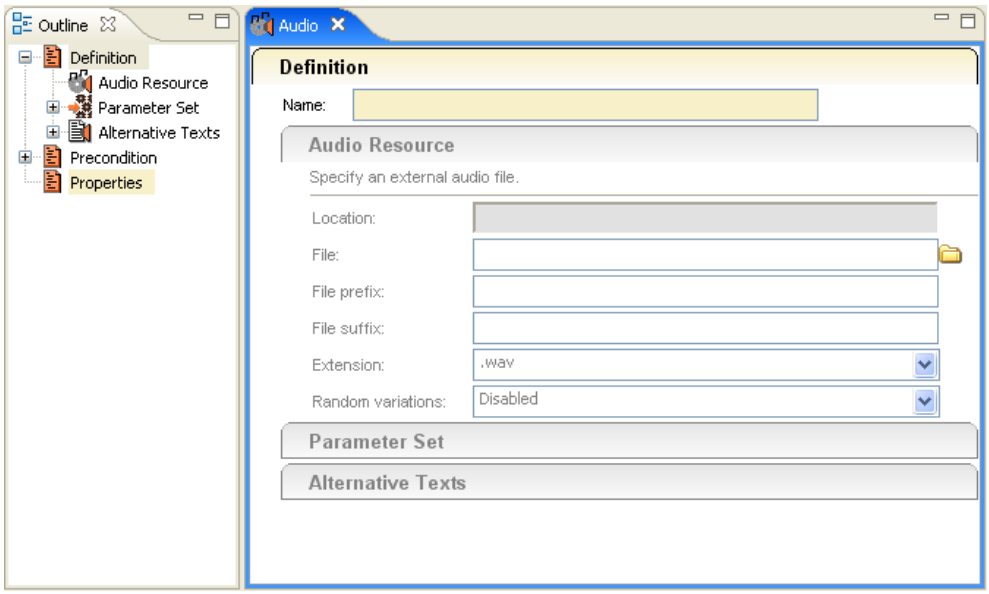
Object		Dialog Flow
		[Music jingle]
		[*Hello and* welcome *to the VoicePortal.*]

Object Definition

The **Definition** of the Audio object provides the following sections:

- Audio Resource
  To specify an external audio file.
- Parameter Set
  To specify an optional set of parameters to be passed to audio generation or streaming engines.
- Alternative Texts
  To specify the alternative text that will be played back with TTS if the audio file is not available or if the output mode is set to TTS:Audio.

Copyright © 2001-2011 Voxeo Germany GmbH



For further details regarding additional object configuration refer to *Precondition* and *Properties* in this *Object Reference*.

Audio Resource

The **Audio Resource** section provides the following properties:

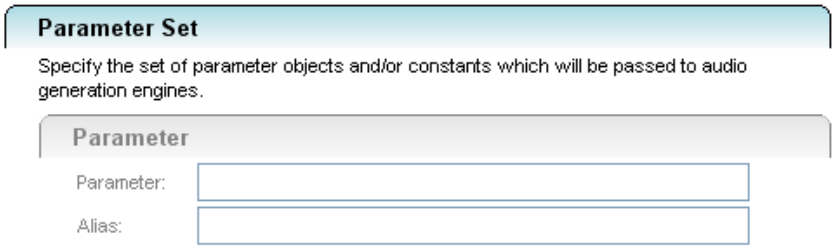
Property	Description
Location	Optional parameter to specify the location (using a Resource Locator object) of the audio file to be used.
File	Defines the file name of the audio file to be used. The name of the file can either be typed in or selected through a file browser by clicking the **Browse** button to the right of the **File** field. The filename can also be provided dynamically during call time by assigning a Variable, Expression, Script, or Layer object. In Desktop for Web the **Browse** button will only be displayed if the Resource Locator object is available and includes a physical path definition. Moreover the audio file can be played by clicking the **Play** button to the right of the **File** field. This will start the default audio player associated with Internet Explorer and play the audio file in this player. This button will only be displayed if the Resource Locator object is available and includes a URL definition without references to dynamic objects (Variable, Expression, Script, or Layer).
File prefix	Optionally specifies a prefix that is applied to the file name. The file prefix can also be provided dynamically during call time by assigning a Variable, Expression, Script, or Layer object.
File suffix	Optionally specifies a suffix that is applied to the file name. The file suffix can also be provided dynamically during call time by assigning a Variable, Expression, Script, or Layer object.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Random variations	Defines whether randomizing is enabled for the audio file. To enable randomizing set the **Random variations** property to a number between 2 or 99, by selecting an appropriate value from the drop-down list. If randomizing is enabled and different audio files are provided with the specified file name extended by different numbers (e.g. *welcome1.wav*, *welcome2.wav*, *welcome3.wav*), the server will randomly pick one of the available files with a random file extension between 1 and the selected number whenever the Audio object is processed. True randomizing ensures that all available audio files are played before a repetition occurs. By default, the **Random variations** property is set to *Disabled* indicating that no randomization is used. For more information on randomization see the Output object.
Extension	Specifies the audio file extension. Instead of selecting one of the provided file extensions from the drop-down list, the extension can also be supplied by using a Variable, Expression, Script, or Layer object. The complete filename for the audio file is generated at call time according to the following formula: `[prefix] + file + [suffix] + [random number] + [extension]`

Parameter Set

The **Parameter Set** section is optional and relevant only for Audio objects that refer to external audio generation engines. Use it to define any number of parameters, all of which will be added to the final URL as CGI parameters, in order to control the engine if required.



This section provides the following properties:

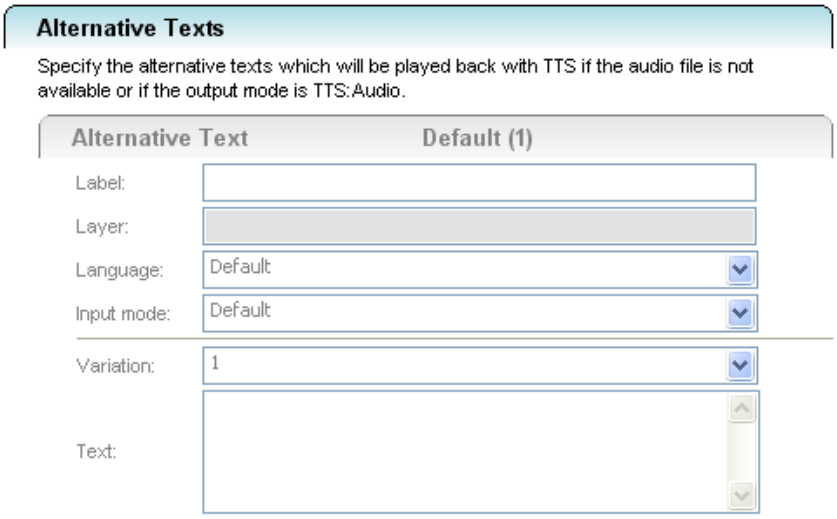
Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Parameter	Specifies the value of the parameter. Possible parameter types are Variable, Collection, Expression, Layer, Resource Locator or Script objects, and constants. When using a Resource Locator object as a parameter, its URL definition is passed to the script. The field can be left empty to denote an empty value.
Alias	Optionally specifies the name of the parameter. If left empty, the reference ID of the object defined in the **Parameter** field will be used as the name. This means in particular that for constant values an alias needs to be defined (as they do not have a reference ID). Constant parameters without a defined alias are ignored.

The parameters will be added to the final URL in a CGI-string like fashion.

Alternative Texts

The **Alternative Texts** section optionally defines the text that will be played back to the caller if the audio file is not in the correct format, not available (e.g. the corresponding file server is down or the network connection is not available) or *Output mode* is set to *TTS:Audio* (for more information on *Output mode*, see Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*).



The **Alternative Texts** section contains items to define the various texts for prompt variations, in case the Audio object represents more than one file. The text entered into the **Text** field must not contain special characters such as quotation marks, apostrophes, ampersands etc. Also avoid using angle brackets unless you want to include SSML tags into your text. The alternative text can be typed in or can be set at call time by a Variable, Expression, Script, or Layer object.

In addition to the **Text** field, the following properties can be configured:

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Label	Optional parameter to identify the alternative text item in a list.
Language	Audio objects can represent prompts in multiple languages through dynamic resource locators. The **Language** drop-down list allows you to specify a language for a prompt, so that prompts of different languages can be incorporated into the Audio object. When dealing with monolingual applications, this field can be ignored.
Layer	Associates the alternative text item with a custom layer. This item is only eligible for playback if the layer evaluates to *true* at call time. If you reference Audio objects in different Output object items using different layer conditions, there is no need to also define the same layers in the Audio object itself. If, on the other hand, different outputs are achieved by using a Layer (or Variable, Expression, Script) object within the resource locator definition of the Audio object, you can define different alternative texts by associating items with the corresponding layer condition.
Variation	If randomizing is enabled, this field allows you to associate a text with a specific prompt variation number. If the Audio object only refers to a single file, this can be left at the default value. Do not define multiple alternative text items with the same variation number (and the other settings being identical), as this may lead to unexpected behavior at call time. If you want to use the same alternative text for all randomized audio files, you only need to define an alternative text for variation number 1.
Input mode	Associates the alternative text item with a specific input mode. Leave this property at the default value if your application does not distinguish between voice and DTMF input in the wording of your application prompts. Otherwise, tag the item with the appropriate mode.

Note: If the audio file is not available, not in the right format, or *Output mode* is set to *TTS:Audio*, the alternative text will be synthesized by the TTS (text-to-speech) system of the underlying media platform. If the media platform does not support TTS the caller will not hear anything.

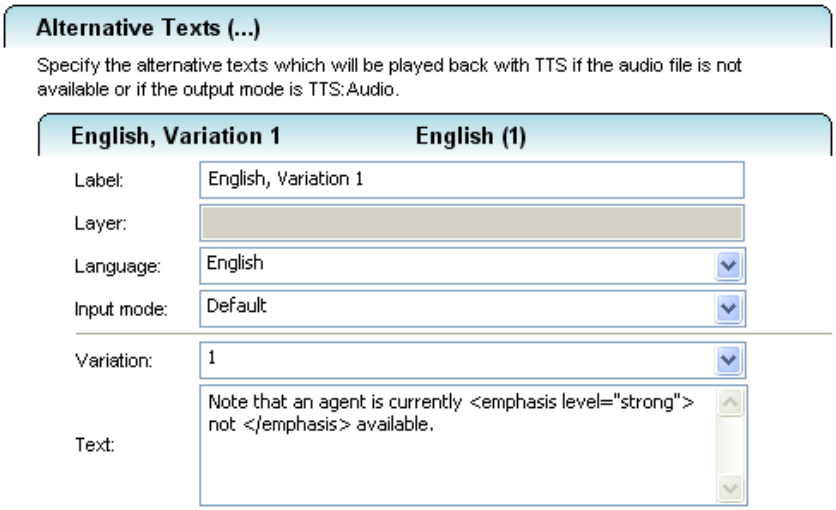
For further details on how to use an Audio object, see the Output object in this *Object Reference*.

SSML

The Speech Synthesis Markup Language (SSML), developed by the W3C Voice Browser Working Group, allows adding meta information to text that is supposed to be read out through a speech synthesis system. By using SSML, you have better control over how the TTS engine synthesizes speech. It allows you to define voice-related parameters like volume, speed, pitch, emphasis, pronunciation, etc. For more information on SSML, refer to the specification at the W3C website, or to chapter 4.1.1 of the VoiceXML 2.0 specification.

Copyright © 2001-2011 Voxeo Germany GmbH

SSML markup can be included in any alternative text item, as well as in Output items. The server passes the extra code to the media platform without changes. The following example shows how to use SSML to tell the TTS engine to put emphasis on a specific word within a given prompt:



Caution: Since the markup you enclose in the alternative text item definition is passed to the platform as it is, make sure to provide the correct syntax required by SSML. Note that it is up to the platform or TTS engine, respectively, if and how SSML is interpreted. If a TTS engine does not support SSML, it might decide to read the XML tags out aloud.

VoiceObjectsXML Definition

The Audio object is represented by the VoiceObjectsXML element `<audio>`. It has six attributes and three groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<audio>` element uses the embedded `<altText>` element.

Audio

Attributes

- **`location`**
  Defines the location from where the audio file is to be retrieved. Must be a reference to a Resource Locator object.
- **`file`**
  The file name of the audio file, without extension. Can be a constant name, or a reference to a Variable, Expression, or Script object.
- **`prefix`**
  A prefix to be used together with the audio file name.
- **`suffix`**
  A suffix to be used together with the audio file name.

Copyright © 2001-2011 Voxeo Germany GmbH

- **random**
  Indicates whether randomization should be used. Can be either *disabled* or an integer >= 2 specifying the number of versions in which the file is available. If not specified, defaults to *disabled*.
  For further information on randomization see Random Variations above.

- **audioFileExtension**
  Specifies the file extension for the audio file as either a constant value or a reference to a Variable, Expression, or Script object. Legal constant values are *none, aif, aiff, alaw, alw, au, dwd, ivc, mp3, snd, ulaw, ulw, voc, vox, wav*. If not specified, defaults to *wav*.

  The complete filename for the audio file is generated at call time according to the following formula:

```
[prefix] + file + [suffix] + [random number] + [extension]
```

Children

- **<expression usage="precondition">** or
  <variable usage="precondition"> or
  <collection usage="precondition"> or
  <script usage="precondition">
  Defines the precondition for the Audio object.

- **?<parameterSet>**
  Defines the parameter set.

- ***<altText>**
  Defines the corresponding alternative text(s).

Example

```
<audio location="#Prompt locator" file="welcome"
audioFileExtension="wav"/>

<audio location="#Prompt locator" file="nomatch"
audioFileExtension="wav" random="5"/>
```

ParameterSet

Children

- **+<item>**
  Defines the list of entries in the parameter set. The use of the `alias` attribute is optional when the `object` attribute is used, and required when the `value` attribute is used.

Example

```
<parameterSet>
  <item alias="prompt" object="#Prompt"/>
  <item alias="speed" value="0.5"/>
</parameterSet>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Item

Attributes

- **`object`**
  Defines a reference to an object. If this attribute is defined, the `value` attribute must not be defined.
- **`value`**
  Defines a constant value. If this attribute is defined, the `object` attribute must not be defined.
- **`alias`**
  Defines the parameter alias.

Example

```
<item alias="prompt" object="#Prompt"/>
```

AltText

The `<altText>` element defines the alternative texts to be used for the Audio object. It may only occur embedded inside an `<audio>` element.

The `<altText>` element has six attributes and contains text as a child.

Attributes

- **`label`**
  A text string providing a name for the alternative text.
- **`random`**
  Defines the random index to which this alternative text belongs. An integer >= 1. If not specified, defaults to 1.
- **`language`**
  Defines the language for which this alternative text is valid. Can be *default* or a valid language code (e.g. *de-DE*, *en-US*, etc.). If not specified, defaults to *default*. Appendix A – *Language Codes* contains a list of all language codes available in VoiceObjects together with the respective language they represent.
- **`inputMode`**
  Defines the input mode for which this alternative text is valid. Can be *default*, *voice*, *dtmf*, or *voicedtmf*. If not specified, defaults to *default*.
- **`layer`**
  Defines the layer for the alternative text. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form "Layer=State" or "Layer!=State" where "State" is the label of a state for the layer "Layer".
- **`object`**
  Optional reference to an Expression, Variable, or Script object that defines the alternative text.

Copyright © 2001-2011 Voxeo Germany GmbH

Children

- **CDATA**
  The static alternative text itself. Note that the text may not contain special characters such as apostrophes, quotation marks, etc. It may, however, contain XML markup to be used e.g. in combination with SSML.

Example

```
<altText language="de-DE">
  Bitte geben Sie Ihre Telefonnummer ein.
</altText>

<altText random="1"> Sorry? </altText>
<altText random="2"> Once more, please. </altText>
<altText random="3"> I didn't quite get that. </altText>
```

Object Interoperability

The following table contains all object types that can reference an Audio object:

Icon	Object Name	Use Case Example
	Output	An Audio object can be referenced within an Output object definition. In addition, an Audio object can be referenced in all objects that provide Output items.
	List	An Audio object can be referenced within the Content Formatting definition of a List object, to play back pre-recorded audio as part of the List content.
	Connector	An Audio object can be linked within a Connector object as the Wait Loop Audio.
	Database	An Audio object can be linked within a Database object as the Wait Loop Audio.
	Script	An Audio object can be linked within a Script object as the Wait Loop Audio.
	OSDM	An Audio object can be linked in the parameter set within any OSDM object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of an Audio object should be derived from the name of the Output object it is used in. When used in multiple outputs, its name should provide information about what is being played. The table below lists three examples:

Copyright © 2001-2011 Voxeo Germany GmbH

Name	Description
Get claim reference number, Help 1	Plays first level help for the Input object *Get claim reference number*
Generic No Input 1	Contains five alternatives for a generic first level No Input prompt (e.g. *Sorry?*)
Main menu earcon	Short jingle played every time the caller returns to the main menu

Copyright © 2001-2011 Voxeo Germany GmbH

Business Task

Overview

The Business Task object is used to define a business view on top of the dialog definition of an application. Business tasks can structure an application in terms of tasks that a caller can accomplish, such as logging into an account or doing a transaction. Information on the processing of a task will automatically be logged to Infostore, the logging component of VoiceObjects Server, including success and failure rates, the time that a task was active, data that was gathered during the task, recognition rates, etc. Using VoiceObjects Analyzer, powerful reports can be generated to understand how successful callers are in using your applications. Since business tasks are mainly implemented for logging and reporting purposes, Infostore must be licensed to make use of this feature.
Business tasks have no influence on the dialog flow or the interaction with the caller.

The Object Definition below covers the configuration of the Business Task object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Business Task object belongs to the object category Business Tasks.

Dialog Flow Scenario

The following example shows how a business task *Life insurance information* is started and finished in a dialog flow using Expression objects.

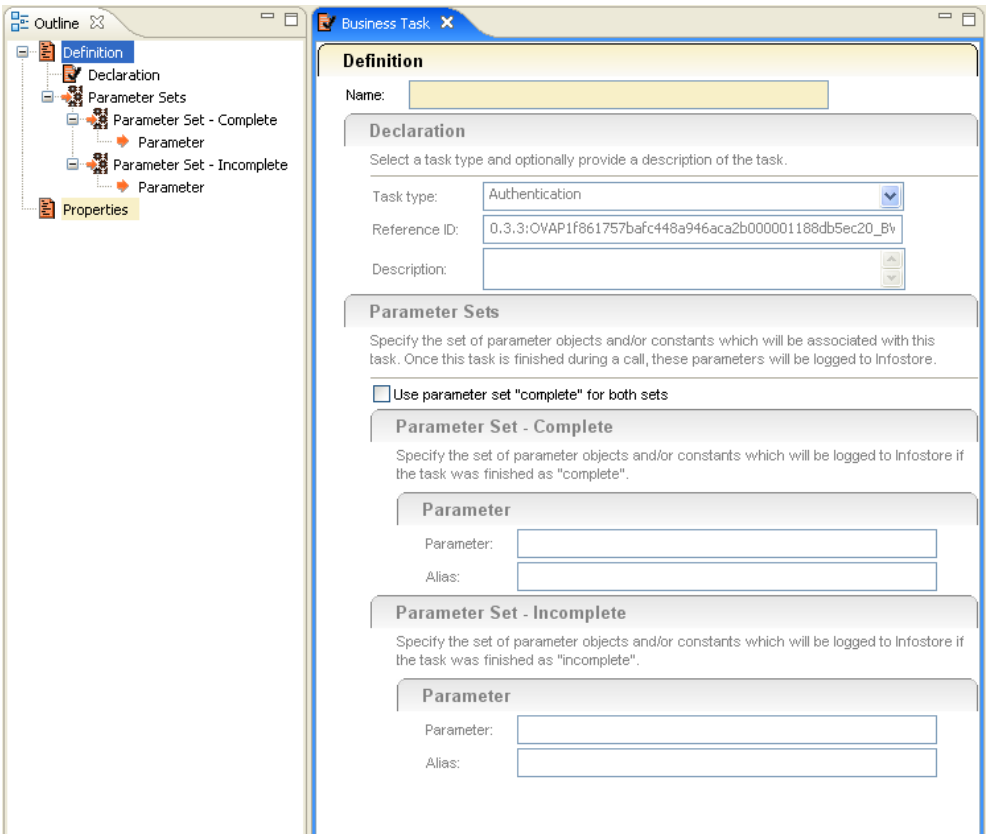
Object - Caller	Dialog Flow
Start task *Life insurance information*	[Starts business task *Life insurance information*]
	What is your gender?
Caller	*I am male.*
	And what is your age?
Caller	*Thirty-one.*
	[Calculates life insurance fee]
	The fee for a thirty-one year old male starts at just 92 dollars.
Finish task *Life insurance information*	[Finishes business task *Life insurance information* as *complete*]

Copyright © 2001-2011 Voxeo Germany GmbH

Object Definition

The **Definition** of the Business Task object provides the following sections:

- Declaration
  To specify the task type, optionally change the reference ID and provide a short description.
- Parameter Sets
  To define parameter sets to be logged to Infostore in case the task was finished either as *complete* or as *incomplete*.



For further details regarding additional object configuration refer to Properties in this *Object Reference*.

Note: A Business Task object can be disabled. In this case, none of the processing described below will take place.

Copyright © 2001-2011 Voxeo Germany GmbH

Declaration

In the **Declaration** section, the task type is specified that categorizes the business task. One of the following types can be selected from the drop-down list in the **Task type** field:

Task type	Description
Authentication	Any kind of authentication, identification or verification task, such as entering an account number and PIN at the beginning of a call
Information	Any kind of information that is presented to the caller, such as playing back the account balance in a phone banking application
Notification	Any kind of external notification sent to the caller, such as email, SMS, MMS, etc.
Routing	Any kind of routing task, such as transferring the caller to an agent
Transaction	Any kind of transaction task, such as booking a flight, transferring money, ordering a product, changing credit card details, etc.
Other	A task that does not fit in any of the other task types

Using the function TASKREFID() through an Expression object, the reference ID of the currently active task can be retrieved at call time. Since readable IDs are usually preferred, the reference ID of the Business Task object can be modified in the **Reference ID** field.

A brief description of the task can be entered in the **Description** field. This description corresponds to the short description for the Business Task object (for further information refer to Properties in this *Object Reference*).

Parameter Sets

Use the **Parameter Sets** section to associate data with the task, which will be logged to Infostore when the task is finished. Specifying parameter sets is optional. Possible parameter types are Variable, Collection, Expression, Layer, Resource Locator, or Script objects, and constants. When using a Resource Locator object as a parameter, its URL definition is used.

The **Alias** field may be used to define the name under which the parameter will be logged to Infostore. Using the **Alias** field is optional. If left empty, the reference ID of the object in the **Parameter** field will be used. This means in particular that for constant values, an alias needs to be defined (as they do not have a reference ID). Constant parameters without a defined alias are ignored.

A distinction is made between tasks that are finished as *complete* and those that are finished as *incomplete* (this is described in the paragraph on Processing). All parameters defined in the **Parameter Set – Complete** subsection are used in case the task is finished as *complete*. All parameters defined in the **Parameter Set – Incomplete** subsection are used in case the task is finished as *incomplete*. For your convenience, select the option **Use parameter set "complete" for both sets** to re-use the parameters defined in the complete set also for the incomplete case. If

Copyright © 2001-2011 Voxeo Germany GmbH

parameters are defined in the incomplete set and this option is selected, these parameters will be ignored and the ones from the complete set will be used instead.

Processing

This section explains how business tasks are processed in a call, introduces some terminology and describes the type of data that will be logged to Infostore.

Start business tasks

A business task can be started (and finished) any number of times during a call.

Business tasks are added to the dialog flow by placing an Expression object with the STARTTASK function and the Business Task object as its argument at the desired position. When the caller passes this object at call time, the task is started.

Note that this is the only way to start a business task; you cannot add a Business Task object itself to the dialog flow.

Finish business tasks

A task can be finished in two different ways, either manually or automatically. When a task is finished, data on the processing of the task is logged to Infostore by the server.

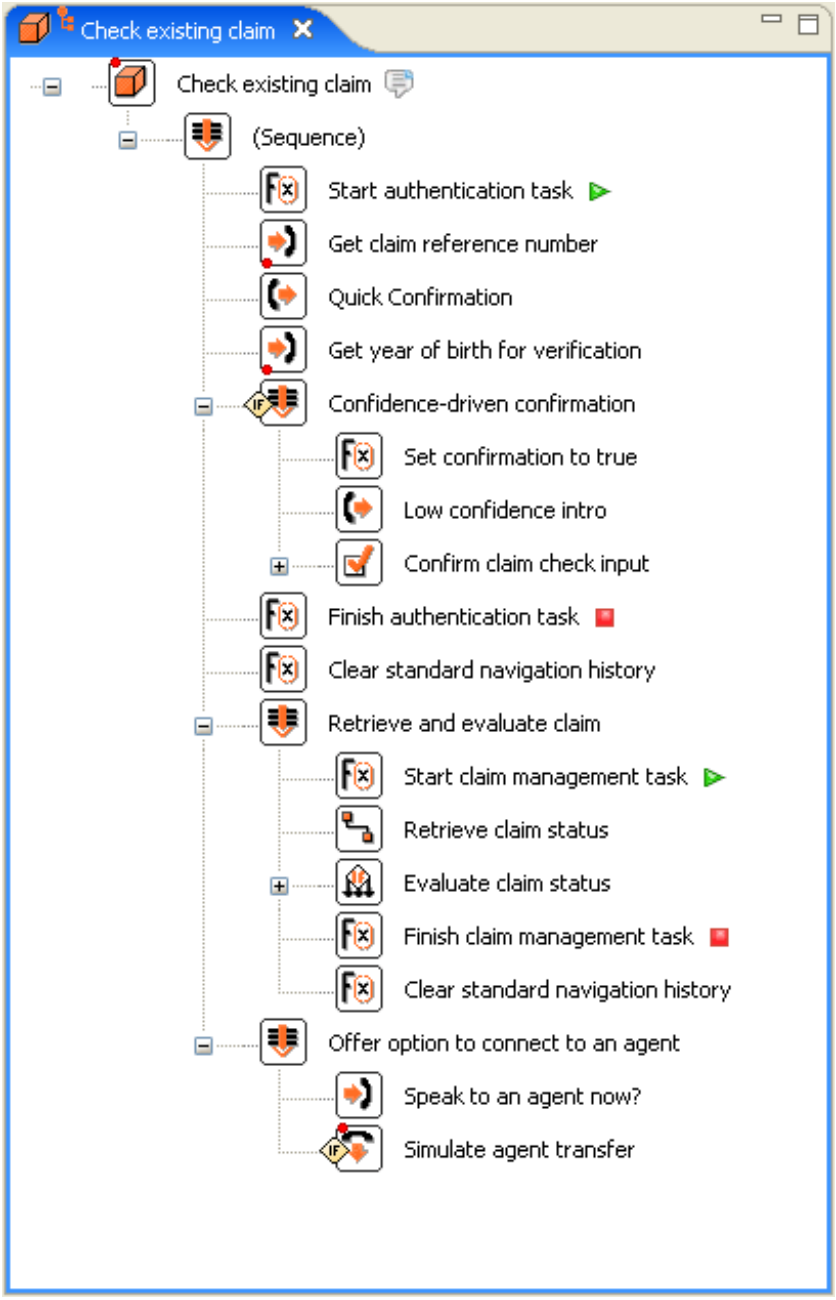
Finish a task manually

The end of a task is marked manually by placing an Expression object with the FINISHTASK function at the desired position in the dialog flow. As first argument of the function, link the Business Task object to be finished. The second argument controls the status you want to assign to the task when finishing it. Leave it empty or define it as *complete* to mark the status as complete when the caller passes the Expression object at call time, which should be the default when finishing a task with the FINISHTASK function. See Completion and status of a task below for more information on the status.

Instead of linking a Business Task object as first argument, you can also enter *active* to finish the currently active task, or *all* to finish all tasks. See Handle more than one active task below for more details on processing several tasks simultaneously.

Expression objects with the functions STARTTASK and FINISHTASK are displayed in the dialog flow with a start icon and a stop icon respectively, to highlight the use of business tasks in a dialog flow. The following example is taken from the *Check existing claim* sub module within the *Prime Insurance* sample application. It shows the start and end points of a business task to authenticate the caller and a task to manage claims, respectively:

Copyright © 2001-2011 Voxeo Germany GmbH



Note: When the FINISHTASK function is processed for a business task that is not started yet, or already finished, the function will be ignored.

Finish a task automatically

Instead of finishing a task manually through Expression objects, the server can finish a task *automatically* depending on certain conditions at call time. This reduces workload for the developer, as the server can handle certain scenarios related to business task processing itself, without requiring explicit definitions by the developer.

When the server finishes a task automatically, it will be marked as *incomplete* by default. For a definition of this term, see Completion and status of a task below.

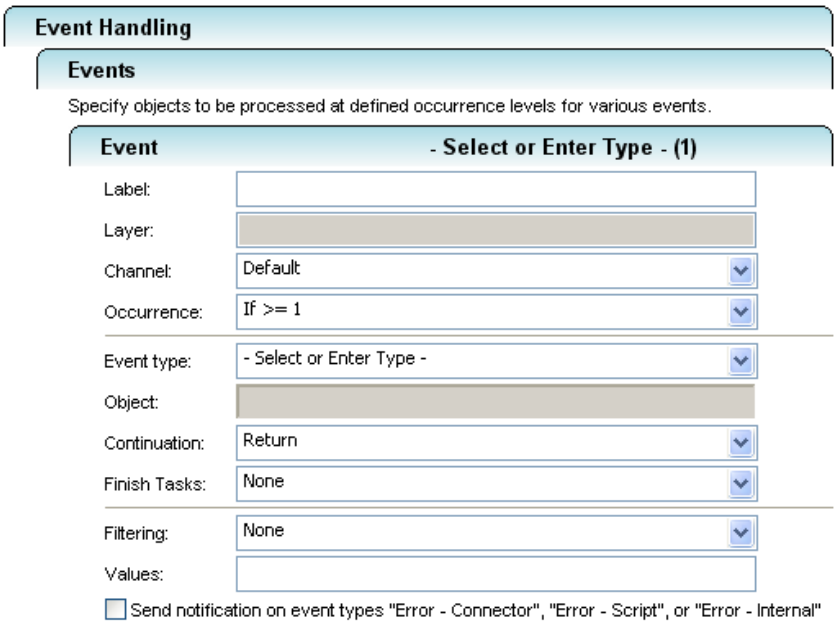
Copyright © 2001-2011 Voxeo Germany GmbH

A task is finished automatically

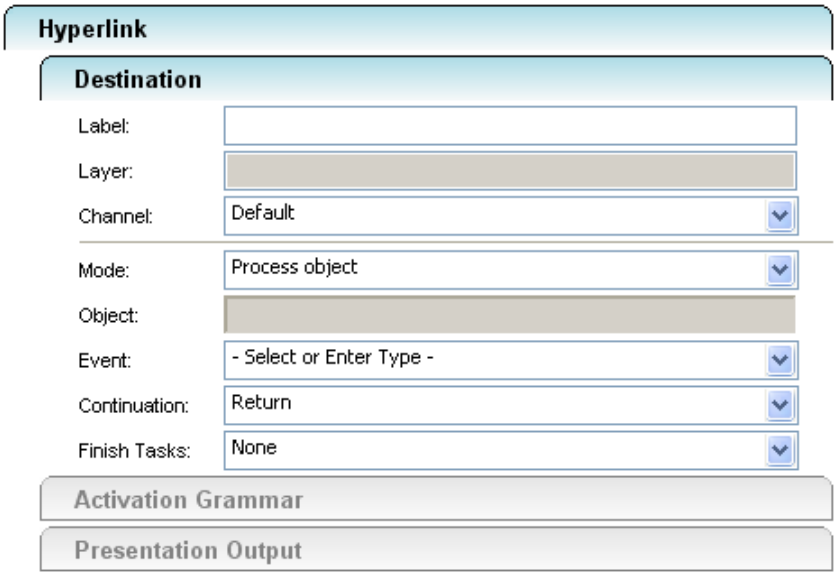
- when the call ends (dialog timeout, no further objects to process, uncaught runtime errors) before the task has been finished manually through the FINISHTASK function. In this case, any dialog end processing objects defined in modules are processed first, and the task is finished at the very end of the call, with the values the parameter sets have at that point in time.
- when an Expression object with the STARTTASK function is processed again for a business task that has not been finished yet. The current instance of the task first gets finished, and a new instance of the same task will then be started again.
- when an event handler, hyperlink or Goto object is used that has the **Finish Tasks** option set to *Active* or *All*.

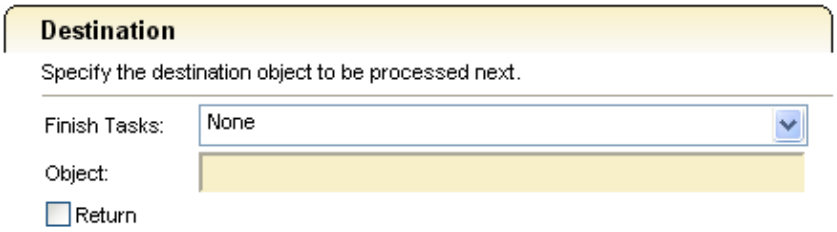
In an event handling item, a custom navigation item, a Hyperlink object, and a Goto object, the **Finish Tasks** option can be set to *Active*, if you want to finish the currently active task, or to *All*, if you want to finish all tasks when the respective object or item is processed. The default is *None*, which means that no task is finished.

The following screenshots show the option in an event handling item, custom navigation item and Goto object, respectively:



Copyright © 2001-2011 Voxeo Germany GmbH





Note: The **Finish Tasks** option will only be processed on a hyperlink if the mode is set to *Process object*. On an event handling item, the option will only be processed if the target is an Exit or Goto object, i.e. it will not be processed if the event handler merely plays a prompt through an Output object.

Tip: It is best practice to set **Finish Tasks** to *All* for a hyperlink that returns to the main menu, as well as for event handlers that exit the call. In general, make sure to consider the **Finish Tasks** option carefully when designing an application.

While the task will always be marked as *incomplete* when finished automatically, the status depends on the processing context.

Completion and status of a task

When a task is finished, either manually or automatically, it will be assigned a completion value and a status. Completion can be either *complete* or *incomplete*. A task can only be marked as *complete* when finished manually through the FINISHTASK function. The second argument must be left empty or set to *complete* in this case. The status of a completed task is also *complete*.

If a task is finished automatically, or using the FINISHTASK function with a value other than the empty string or *complete*, it will be marked *incomplete*. The status then further specifies why the task was incomplete. In case of the FINISHTASK function, the status can be set by the developer. In case of the server finishing a task automatically, the context in which the task was finished determines the status.

Copyright © 2001-2011 Voxeo Germany GmbH

The following table lists each possible status a task can get in case of *incomplete*.

Status	Description	Set automatically…
Back-end error (backEndError)	The task was finished due to an error in a back-end system.	with event handler *Error – Connector*
Business logic (businessLogic)	The task was finished due to requirements or specifications of the business logic.	n/a
Caller abort (callerAbort)	The task was finished by means of caller interaction.	with event handlers *Caller - Exit*, *Caller - Cancel*, *Caller – Help* with all custom event handlers (custom event type) with the Goto object with the Hyperlink object
Recognition failure (recognitionFailure)	The task was finished due to speech recognition problems.	with event handlers *ASR – Invalid Answer Limit,* *ASR – Max Processing Time*, *ASR – Max Speech Duration*, *ASR – No Input*, *ASR – No Match*, *ASR – No Match (Voice)*, *ASR – No Match (DTMF)*
Task restart (taskRestart)	The task needed to be finished since the same task was started again. **Note:** This status cannot be set manually. Trying to do so using FINISHTASK will result in an error.	when an unfinished task is started again

Copyright © 2001-2011 Voxeo Germany GmbH

Status	Description	Set automatically…
Session termination (sessionTermination)	The task was finished since the dialog session terminated. **Note:** This status cannot be set manually. Trying to do so using FINISHTASK will result in an error.	when a call ends due to dialog timeout or end of call (no more objects to process) with event handlers *Disconnect*, *Disconnect – Hangup*, *Disconnect – Transfer* and with uncaught errors that end the call
Technical error (technicalError)	The task was finished due to a technical error.	with event handlers *Error – Internal*, *Error – Media Platform*, *Error – Script*

The strings in parentheses in the **Status** column denote the values to be used in the FINISHTASK function as second argument. They are the same as the ones used in VoiceObjectsXML, as well as the values returned by the TASKSTATUS function.

The TASKSTATUS function returns the status of the business task provided in the first argument. If a task has never been started in a call, the function returns *notStarted*. If a task has not been finished when calling the function, the string *active* or *inactive* is returned, depending on whether the task is currently active or inactive (for a definition of these terms, see Handle more than one active task below). If a task was started more than once during a call, the status of the last processed task instance is returned.

The function TASKREFID can be used to retrieve the reference ID of the currently active task. If no task is currently active, the function returns an empty string.

Information about the most recently finished task can be retrieved using the LASTTASK function. It provides the task's completion status, duration, occurrence, Reference ID, status, and number of input states.

Handle more than one active task

In most cases it should be sufficient to have one task active at a time. A task is started, some objects are processed, and the task is finished before another task is started. In some cases, it might be desirable to start another task while one task is still active. This is possible by processing two or more STARTTASK functions before finishing a task.

The tasks are then managed on a *stack*. Only one task can be active at a time. If a task B is started while task A is still active, task B will become active (i.e. pushed to the stack) and task A will become inactive. Task B is considered a *child* task of task A, so they should be conceptually related. Usually, task B should be finished (i.e. removed from the stack) prior to task A. In that case, task A becomes active again. If task A is finished before task B (by calling FINISHTASK on task A), then task B will be implicitly finished by the server as *incomplete*, with the same status that has been assigned to task A. In other words, finishing an inactive task will implicitly finish all tasks above the task in the stack. A special column IMPLICIT_FINISH in Infostore marks the implicitly finished tasks.

Copyright © 2001-2011 Voxeo Germany GmbH

Task data logging in Infostore

Once a task has been finished, information on the processing of the task is automatically logged to Infostore. This requires Infostore to be licensed and System DB logging to be enabled at call time. If System DB logging is turned on, information on business tasks will be logged. The coverage filter, which can control the amount of information logged to Infostore, thus also applies to business task logging, if the filter scope includes System DB logging.

The following tables are used in Infostore to store business task-related information.

Lookup tables

Table	Description
VOLDTASK	Stores information on the Business Task objects defined in your application, such as their names, creation and modification timestamps, task types etc. This table will be filled whenever the server loads an application into its cache, i.e. on redeploy and restore operations.
VOLDTASKTYPE	Holds the IDs and localized descriptions of the six available task types: *Authentication*, *Routing*, *Information*, *Notification*, *Transaction*, *Other*.
VOLDTASKCMPL	Holds the IDs and localized descriptions of the two available completion values: *complete* and *incomplete*.
VOLDTASKSTATUS	Holds the IDs and localized descriptions of the eight available task statuses: *Complete*, *Back-end error*, *Business logic*, *Caller abort*, *Recognition failure*, *Task restart*, *Session termination*, *Technical error*.

Fact tables

Table	Description
VOLDTASKSTATS	Stores information on the processing of a task at call time. Including: Start and end timestamp, duration and number of active and inactive phases, task status, recognition rates (No Match, No Input events, average confidence values etc.), number of errors, etc.
VOLDTASKDATA	Stores information on the data associated with a task in the object definition, i.e. parameters gathered during the call. Examples are account numbers, selected menu or list items, answers provided in Input objects, etc. The values will be logged to two columns, one with an alphanumerical, and the other one with a numerical data type. This allows you to do

Copyright © 2001-2011 Voxeo Germany GmbH

Table	Description
	calculations such as building a sum or an average on the data, in case they are numerical.

In addition, the tables VOLDDSSEQ and VOLDRECLOG, which store input state-related data, are enhanced to store the active business task for a given input state, if there is any. The table VOLDDLGSTS holds aggregated business task information for the entire dialog session.

To be able to associate the logs written to Infostore with custom logging implemented in your application, the placeholders @TASK_SID@ and @TASK_COUNTER@ can be used inside a Log object. They will be replaced with the surrogate ID (unique ID used in the VOLDTASK table) and the counter of the currently active business task.

Note: Even if System DB logging is disabled, business tasks are still processed.

Business tasks and standard navigation

Using the standard navigation commands *backward* and *forward*, the caller can step over a start and end point of a business task. A special handling will apply in these cases.

- The information that is maintained for tasks will not be modified by rollbacks (i.e. going *backward* will not roll back timestamps already memorized, or any of the counters that count the number of events per task, etc.)
- When a *backward* command is used that steps over a STARTTASK function, the task that has been started by this function will be automatically finished with status *callerAbort*. As for the parameter set to log (in case one is defined), the state of values that was active before the rollback will be used.
- If a *backward* command steps over a STARTTASK of a task that has already been finished before, nothing will happen, as the task cannot be finished again. This could happen if the caller uses several backward commands after a task has been finished, which would take them over the FINISHTASK and the STARTTASK function.
- When a *backward* command is used that steps over a FINISHTASK function, no special handling will apply, i.e. the task that has been finished will not be restarted or modified in any way.
  Note that this could mean that the caller might finish the task again with a different status or parameter set; this information would then not be logged.

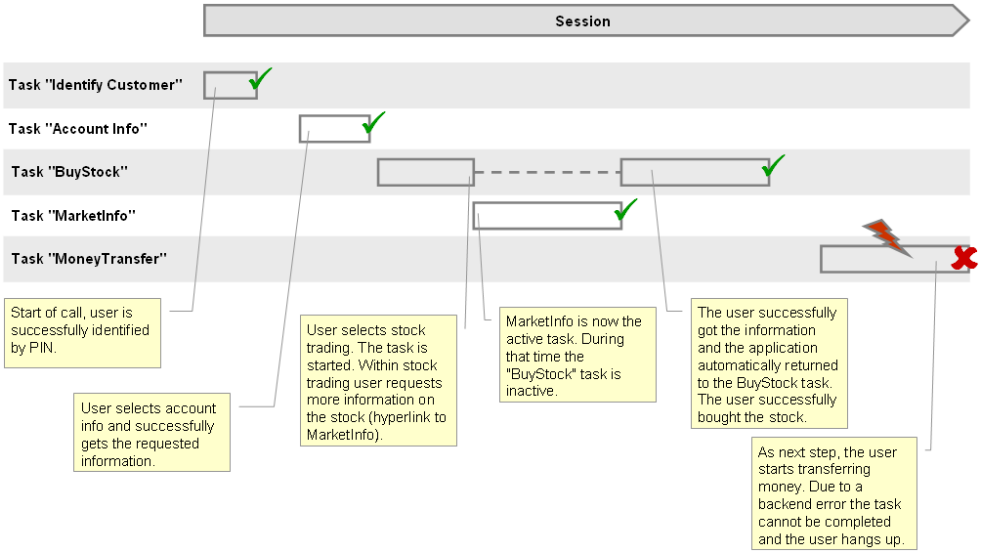
  Tip: To avoid this scenario, use the CLEARNAVIGATIONHISTORY function after a task has been finished. This will make it impossible for the caller to step back over a FINISHTASK function and confuse the statistics on processing tasks. In addition, it will make it impossible to re-run a transaction performed on the back-end.
- When a *forward* command is used that steps over a STARTTASK function, the function will be processed as it is defined, i.e. the task will be started normally as a new task instance.
- When a *forward* command is used that steps over a FINISHTASK function again, the function will be processed as it is defined, i.e. the task will be finished with the status defined.

Copyright © 2001-2011 Voxeo Germany GmbH

- For the *skip* and *repeat* commands, nothing special must be considered. The *repeat* command does not reprocess anything; the *skip* command just leaves the current object and goes on processing the next one.

Example

The following image shows an example dialog flow with several tasks being started and stopped.



- The tasks *Identify Customer* and *Account Info* had been started and successfully finished before any other task was started. Therefore they had 1 active phase, no inactive phase, and are logged with completion and status *complete*.
- The task *BuyStock* had been started, and while it was active the task *MarketInfo* was started. Therefore *BuyStock* became inactive. *MarketInfo* was finished successfully, so that *BuyStock* became active again. This task was then also finished successfully. In the end, *BuyStock* had 2 active and 1 inactive phases, *MarketInfo* had 1 active and no inactive phase, and both are logged with completion and status *complete*.
- The task *MoneyTransfer* was started, but an *Error – Connector* event was triggered. In the Connector object, the corresponding event handler was defined such that it should finish the active task. Therefore *MoneyTransfer* had 1 active phase and is logged with completion *incomplete* and status *Back-end error*.

VoiceObjectsXML Definition

The Business Task object is represented by the VoiceObjectsXML element `<businessTask>`. It has two attributes and two children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<businessTask>` element uses the embedded `<parameterSet>` element.

Copyright © 2001-2011 Voxeo Germany GmbH

BusinessTask

Attributes

- **`taskType`**
  Defines the task type. Can be one of *authentication*, *routing*, *information*, *transaction*, *notification*, or *other*. If not specified, defaults to *authentication*.
- **`useComplete`**
  Defines whether the parameters defined in the complete set will also be used for the incomplete case. Either *true* or *false*. If not specified, defaults to *false*.

Children

- **`<parameterSet usage="complete">`**
  Defines the parameter set to be logged to Infostore in case the task was finished as *complete*.
- **`<parameterSet usage="incomplete">`**
  Defines the parameter set to be logged to Infostore in case the task was finished as *incomplete*.

Example

```
<businessTask taskType="authentication" useComplete="true">
  <parameterSet usage="complete">
    <item alias="accno" object="#AccountNumber"/>
    <item alias="locale" value="de-DE"/>
  </parameterSet>
</expression>
```

ParameterSet

Attributes

- **`usage`**
  Defines the type of parameter set. Can be *complete* or *incomplete*.

Children

- **`+<item>`**
  Defines the list of entries in the parameter set. The use of the `alias` attribute is optional when the `object` attribute is used, and required when the `value` attribute is used.

Example

```
<parameterSet usage="complete">
  <item alias="accno" object="#AccountNumber"/>
  <item alias="locale" value="de-DE"/>
</parameterSet>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Item

Attributes

- **`object`**
  Defines a reference to an object. If this attribute is defined, the `value` attribute must not be defined.
- **`value`**
  Defines a constant value. If this attribute is defined, the `object` attribute must not be defined.
- **`alias`**
  Defines the parameter alias.

Example

```
<item alias="premium" object="#Annual premium"/>
```

Object Interoperability

The Business Task object can only be referenced by an Expression object, which can start and finish business tasks or provide information on the status of a task.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Business Task object should indicate what the task is about. The short description should mention the task type and describe in more detail what kind of task the object represents. The table below lists three examples:

Name	Description
Authentication	Covers the task of authenticating the caller by asking for account number and PIN.
Market info	Plays information on selected stock. Task type is Information.
Transfer money	Covers the task of transferring money to another bank account. Task type is Transaction.

Copyright © 2001-2011 Voxeo Germany GmbH

Case

Overview

The Case object represents a standard CASE or SWITCH CASE construct as in any standard programming or scripting language. It is typically used within the dialog to fork the flow into multiple branches depending on certain business rules, e.g. to switch between different dialog flows depending on the caller's status.

The Object Definition below covers the configuration of the Case object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML, refer to the VoiceObjectsXML Definition paragraph.

The Case object belongs to the object category Logic.

Dialog Flow Scenario

The following dialog flow example shows the use of a Case object in a credit card service portal. It branches the dialog into different subdialogs depending on the status of a caller.

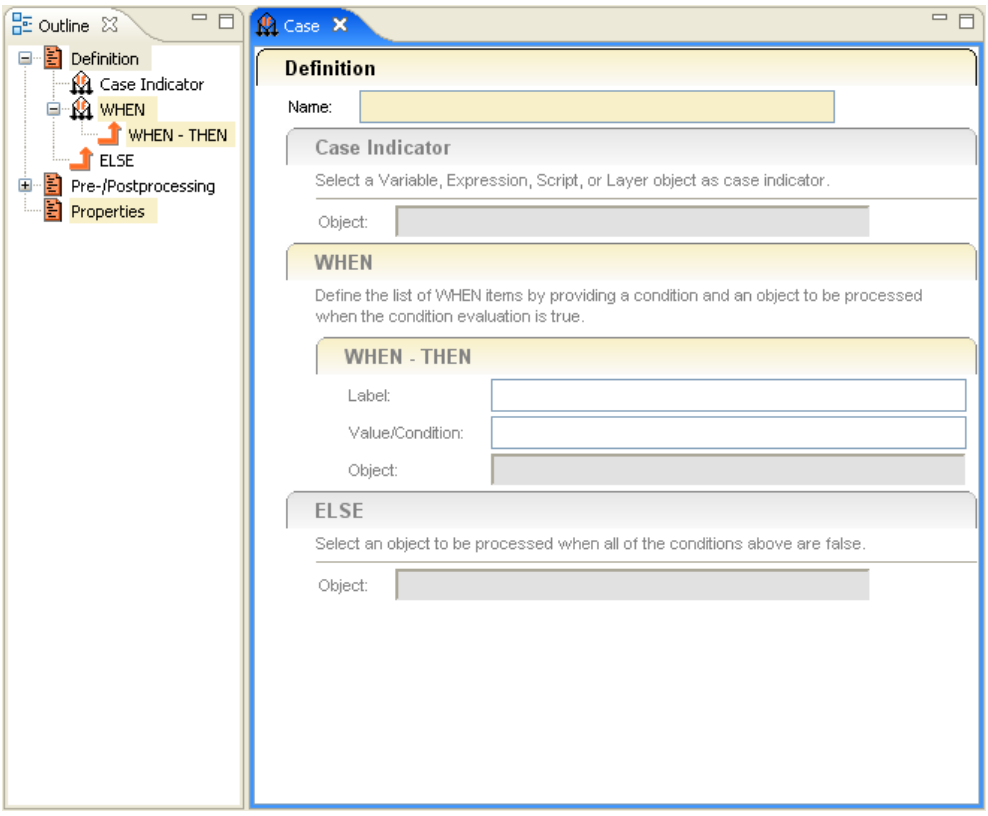
Object – Caller		Dialog Flow
		Retrieve the caller status from the back-end. Can be one of *green_card*, *gold_card*, or *platinum_card*.
		Branch into first dialog flow for status *green_card.*
		Branch into second dialog flow for status *gold_card.*
		Branch into third dialog flow for status *platinum_card.*

Object Definition

The **Definition** of the Case object provides the following sections:

- Case Indicator
  To link a Variable, Expression, Script, or Layer object to compare values to.
- WHEN
  To define the various branching conditions and to link an object to be processed if a condition is true.
- ELSE
  To define an object to be processed in case none of the WHEN items evaluated to *true*.

Copyright © 2001-2011 Voxeo Germany GmbH



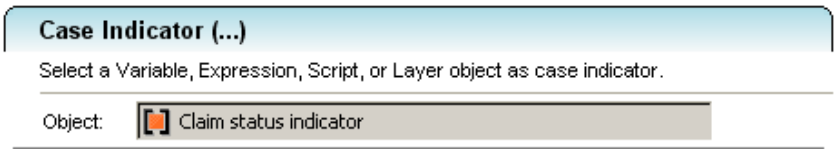
For further details regarding additional object configuration refer to *Pre-/Postprocessing* and *Properties* in this *Object Reference*.

Note: The Case object can be defined in two different **modes**:

- Provide an object as a case indicator and define the WHEN items by comparing the content of this object to some values (static or dynamic).
  Use this mode if the WHEN items all depend on the values of one specific object of your project, typically a Variable or Expression object.
- Specify the WHEN items by explicit condition declarations. In this case, you must not define a case indicator.
  Use this mode if the WHEN items are rather unrelated to each other, and they all depend on distinct conditions.

Case Indicator

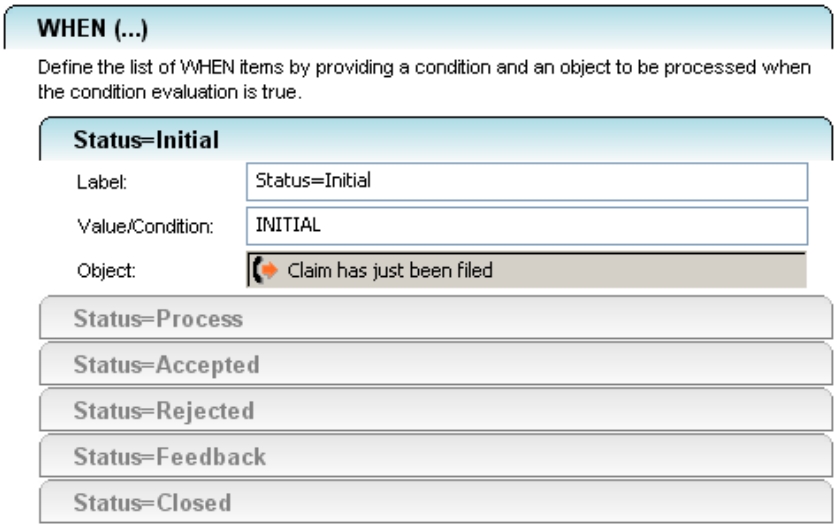
The **Case Indicator** section defines a reference to a Variable, Expression, Script, or Layer object. The content of this object is compared to the values defined in the various WHEN items within the **WHEN** section at call time. Defining the case indicator is only necessary when using the first of the two possible definition modes of the Case object (i.e. comparing a value against a set of values).



Copyright © 2001-2011 Voxeo Germany GmbH

WHEN

The **WHEN** section defines a list of WHEN-THEN items that correspond either to certain values of the case indicator (if one is defined), or to a certain Boolean condition, depending on the mode used for the Case object.



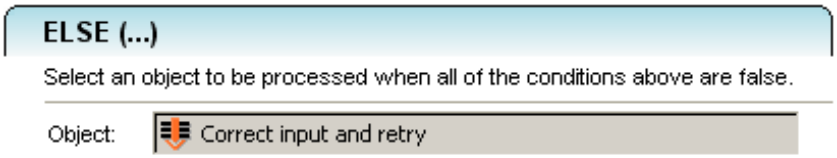
Each WHEN-THEN item provides the following fields:

Property	Description
Label	Optional parameter to identify the WHEN-THEN item in a list. The label is also displayed in the **Dialog Designer**.
Value/Condition	The definition of this field depends on the mode used for the Case object. If a case indicator is defined in the **Case Indicator** section, specify a single value or a comma-separated list of values that are compared at call time to the current value of the object specified as case indicator. You can also link a Variable, Expression, Script, Collection, or Layer object in this field to dynamically determine the comparison values. If no case indicator is defined, link a Variable, Expression, Script, Collection, or Layer object, the evaluation of which will determine if the WHEN-THEN item is processed or not. If the given object evaluates to *true* at call time, the item is processed; if it evaluates to *false*, the item is not processed.
Object	Specifies the object that is to be processed if either the value specified in the **Value/Condition** field equals the value defined in the **CASE** section (if one is defined), or the condition defined for this WHEN-THEN item evaluates to *true*. **Note:** The WHEN-THEN items are evaluated in the order in which they are defined, and only the first one with a matching condition is processed.

Copyright © 2001-2011 Voxeo Germany GmbH

ELSE

The **ELSE** section defines a reference to an object that is processed if none of the WHEN-THEN items defined in the **WHEN** section was activated. The ELSE branch is optional. If it is not defined, and none of the conditions of the WHEN-THEN items evaluate to *true*, the Case object does not perform any action.



VoiceObjectsXML Definition

The Case object is represented by the VoiceObjectsXML element `<case>`. It has one attribute and five groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<case>` element uses the embedded `<caseItem>` and `<else>` elements.

Case

Attributes

- **`caseValue`**
  Defines the case comparison value. Must be a reference to an Expression, Layer, Script, or Variable object.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Case object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Case object.
- **`+<caseItem>`**
  Defines the list of WHEN-THEN items.
- **`<else>`**
  Defines the object to be processed if none of the Case items evaluates to true.
- **`<sequence usage="postprocessing">`**
  Defines the postprocessing sequence for the Case object.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<case caseValue=”#Number of calls”>
  <caseItem label=”Even” values=”2,4,6,8”>
    <output>
      <outputItem>
        <text>You called an even number of times.</text>
      </outputItem>
    </output>
  </caseItem>
  <caseItem label=”Odd” values=”1,3,5,7,9”>
    <output>
      <outputItem>
        <text>You called an odd number of times.</text>
      </outputItem>
    </output>
  </caseItem>
</case>
```

CaseItem

Attributes

- **`label`**
  A text string providing a name for the Case item.
- **`values`**
  Defines either a comma-separated list of constant values, or a reference to an Expression, Script, or Variable object. Either this attribute is defined, or the first child described below is defined, but not both.

Children

- **`<expression usage=”condition”>`** or
  `<variable usage=”condition”>` or
  `<collection usage=”condition”>` or
  `<script usage=”condition”>`
  Defines an embedded object as condition for the Case item.
- One of the following:
  `<module>`, **`<input>`**, **`<output>`**, **`<sequence>`**, **`<menu>`**, **`<confirmation>`**, **`<list>`**, **`<osdm>`**, **`<plugin>`**, **`<connector>`**, **`<script>`**, **`<log>`**, **`<if>`**, **`<case>`**, **`<goto>`**, **`<loop>`**, **`<expression>`**, **`<recording>`**, **`<transfer>`**, **`<pause>`**, **`<exit>`**
  Defines the object to be processed if the condition of this Case item evaluates to true.

Example

```
<caseItem values=”1,3,5”>
  <output link=”#Indicate odd number”/>
</caseItem>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
<caseItem>
  <expression usage="condition" link="#isLeapYear"/>
  <output>
    <outputItem><text>The year is a leap year.</text></outputItem>
  </output>
</caseItem>
```

Else

Children

- One of the following:
  `<module>, <input>, <output>, <sequence>, <menu>, <confirmation>, <list>, <osdm>, <plugin>, <connector>, <script>, <log>, <if>, <case>, <goto>, <loop>, <expression>, <recording>, <transfer>, <pause>, <exit>`
  Defines the object to be processed in the "else" case.

Example

```
<else>
  <output>
    <outputItem><text>The year is not a leap year.</text></outputItem>
  </output>
</else>
```

Object Interoperability

The following table contains all object types that can reference a Case object:

Icon	Object Name	Use Case Example
	Module	A Case object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, a Case object is used within a Sequence object in a dialog flow.
	Menu	A Case object can be used as the destination object within a Menu item.
	Confirmation	A Case object can be used as the destination object within a Correction item inside a Confirmation object.
	If	A Case object can be used within either the THEN or the ELSE item of an If object.
	Case	A Case object can be used in any WHEN item of other Case objects.
	Loop	A Case object can be used within the embedded sequence of a Loop object.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Object Name	Use Case Example
	Goto	A Case object can be referenced via a Goto object.
	Hyperlink	A Case object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Case object should indicate the condition that is evaluated. The short description should contain a brief explanation of the types of cases that may occur. The long description should list all the various cases (WHEN items) that may occur. The table below lists three examples:

Name	Description
 Switch on caller status	Switches to different dialog flows depending on whether the caller is green card, gold card, or platinum card member.
 Offer special rebate	Offers various rebates to the caller depending on the insurance fee retrieved from the DB.
 Switch on caller age	Switches to different dialog flows depending on whether the caller is under 18, between 18 and 21, or over 21.

Copyright © 2001-2011 Voxeo Germany GmbH

Collection

Overview

The Collection object is a set of data values that can be accessed as cells or elements in a two-dimensional data structure, similar to a spreadsheet or a table in a relational database. It is typically used to store the result of a database query or the access result of any other back-end system (e.g. a Web site or a legacy system). The status and the data values of a Collection object are managed within the dialog context of each call. The content of a Collection object is often dynamically populated at call time via a Connector object, which handles the connectivity and communication to any third-party back-end system. It is also used in the List object as the container for the data to be read out.

The content of a Collection object itself is represented through a generic XML data structure using the elements `<root>`, `<row>` and `<col>` and has the following structure:

```
<root>
    <row>
        <col name=”quote”>13.7</col>
    </row>
</root>
```

Generic XML Data Structure

The example above describes a single data cell with the column name “quote” and the assigned value “13.7”.

- **`<root>`** – The `<root>` element is the wrapping XML element for the entire Collection object structure. Only one `<root>` element is allowed.
- **`<row>`** – The `<row>` element forms a block of an arbitrary number of columns represented by the `<col>` element. The order of rows is relevant to the data structure and will always be processed top down, with the numbering starting at 1. The only valid parent element of `<row>` is `<root>`.
- **`<col>`** – The element `<col>` provides the attribute *name* which identifies a column uniquely within a row. Multiple columns of the same name are not allowed. When having multiple rows, the same set of column names should be used in each row to form a homogeneous two-dimensional data structure as presented below with the sample spreadsheet. The only valid parent element of `<col>` is `<row>`.

The following sample spreadsheet containing two columns *A* and *B* and two rows *1* and *2* is outlined below in the corresponding XML data structure of the Collection object:

	A	B
1	Value A1	Value B1
2	Value A2	Value B2

Sample Spreadsheet

Copyright © 2001-2011 Voxeo Germany GmbH

```
<root>
    <row>
        <col name="A">Value A1</col>
        <col name="B">Value B1</col>
    </row>
    <row>
        <col name="A">Value A2</col>
        <col name="B">Value B2</col>
    </row>
</root>
```

XML Data of the Spreadsheet

Note: There are several functions available that allow reading out or manipulating values from a collection. They are categorized as *collection functions* in the Expression object. In addition, the XPath function can be utilized to analyze collection values.

The Object Definition below covers the configuration of the Collection object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML, refer to the VoiceObjectsXML Definition paragraph.

The Collection object belongs to the object category Logic.

Dialog Flow Scenario

The dialog flow scenario demonstrates a List object that plays the content of a Collection object to the caller as part of a Traffic Info service.

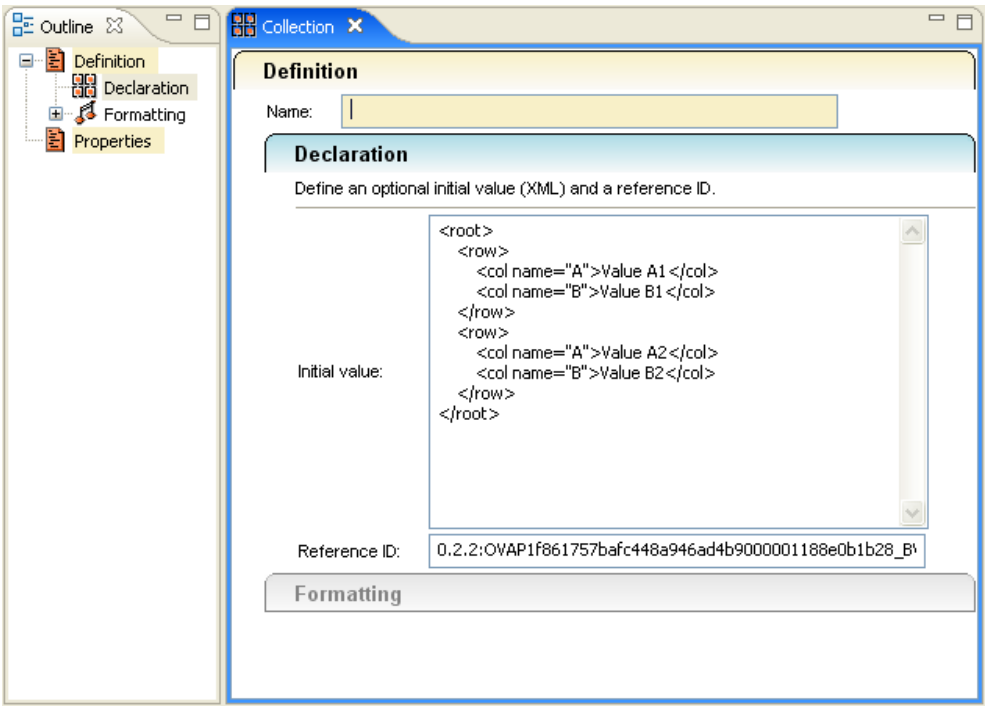
Object		Dialog Flow
		[*This is the list of current traffic reports.*]
		*<Traffic message 1>*
		<Traffic message 2>
		…
		…
		<Traffic message N>
		[*End of the available traffic messages..*]

Object Definition

The **Definition** of the Collection object provides the following sections:

- Declaration
  To specify the initial value and reference ID.
- Formatting
  To define formatting instructions for playback within an Output object.

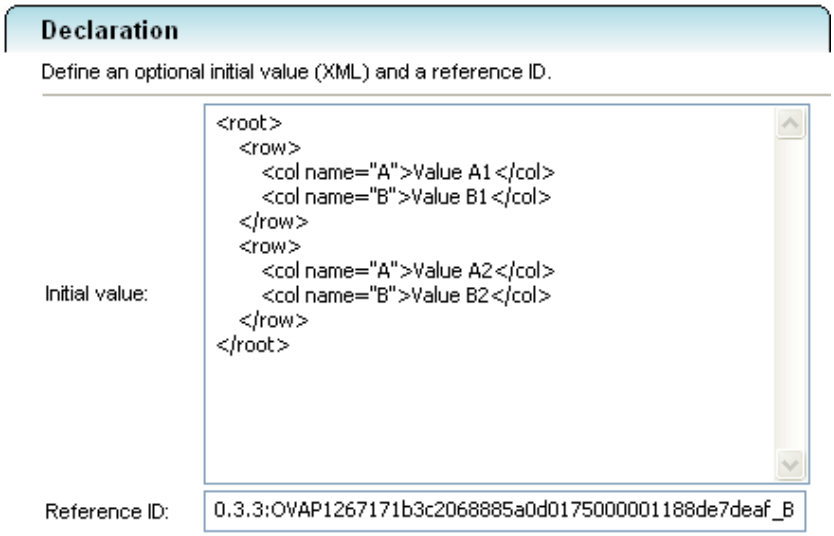
Copyright © 2001-2011 Voxeo Germany GmbH



For further details regarding the **Properties** section refer to *Properties* in this *Object Reference*.

Declaration

Within the **Declaration** section an optional initial value for the Collection object can be defined. For a new Collection object, a sample initial value is provided as shown below.



The sample value can be modified as desired, or removed altogether if an empty initial value (`<root> </root>` or `<root/>`) is desired.

Copyright © 2001-2011 Voxeo Germany GmbH

The initial value can also be set dynamically at call time by referencing a Variable, Expression, Script, Layer, or another Collection object.

You can use the system functions INIT() and REFRESH() to initialize a collection to the initial value again at a later point in the dialog. REFRESH() re-evaluates the initial value in the current dialog context, whereas INIT() resets the collection to the value it got when it was first initialized. For static initial values, both functions have the same effect.

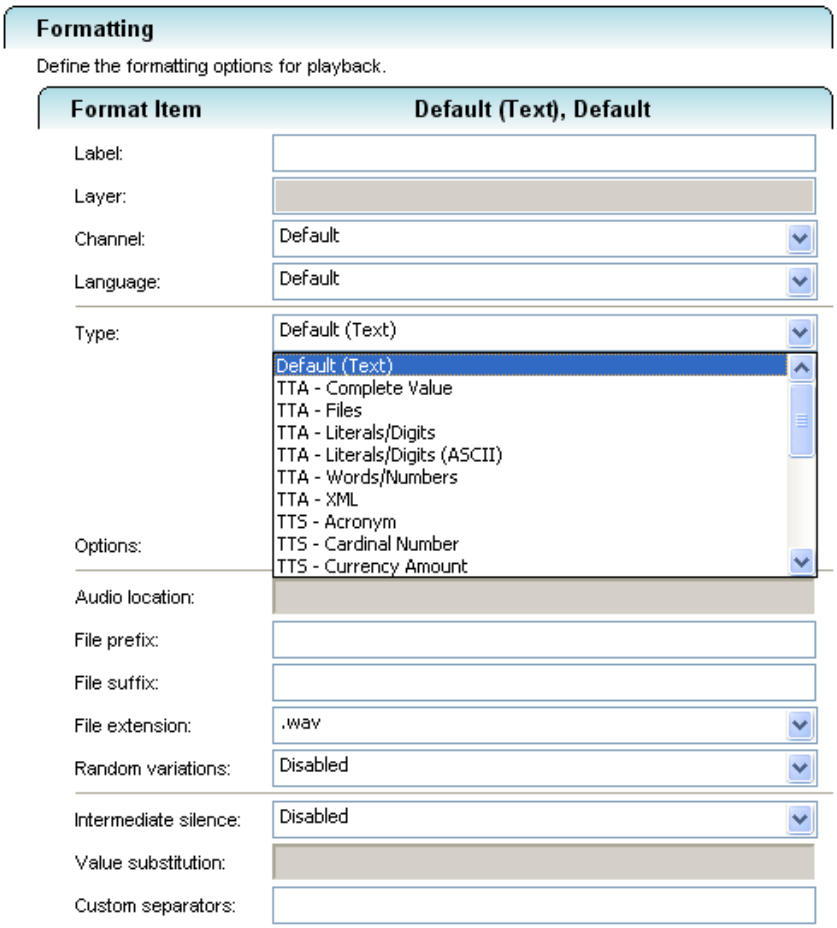
Note: Make sure that the Collection object itself is not part of its own initial value definition (e.g. by being part of the parameter set passed to a Script object that computes the initial value), as this will lead to infinite loops.

The **Reference ID** field can be used to identify and reference a Collection object outside of the VoiceObjects framework, e.g. when using the Collection object in conjunction with a Connector or a VoiceXML Plug-In. In these cases, the automatically generated reference ID should be replaced by a more readable name.

Note that the reference ID may not contain blanks or other special characters except hyphens and underscores. It is not case-sensitive and limited to 40 characters in length. No two objects within one project may have the same reference ID.

Formatting

The **Formatting** section defines how the content of the Collection object is presented to the caller when it is used within an Output object.



Copyright © 2001-2011 Voxeo Germany GmbH

Two different categories of formatting types are offered, which are only relevant for the voice and video channel:

- **Text-To-Audio (TTA)**
  The object content is played back to the caller through the concatenation of prerecorded audio files.
- **Text-To-Speech (TTS)**
  The object content is played back to the caller through text-to-speech synthesis provided by the media platform.

The formatting instructions can be defined as an embedded definition as part of the Collection object, or can be linked to an autonomous Format object. Formatting is applied individually to all entries in the collection, which are read out row-by-row and column-by-column within a row.

To define formatting types for the text and Web channel, use the Formatting bus as described in Appendix B – *How to Use the Formatting Bus* in the *Administration Guide*.

For further details on formatting capabilities refer to the Format object.

VoiceObjectsXML Definition

The Collection object is represented by the VoiceObjectsXML element `<collection>`. It has one attribute and two children.

In addition, the element has the standard attributes described in the *XDK Guide*.



Caution: The `name` attribute is required for Collection objects.

The attribute `referenceID` can be used to identify and reference a Collection object outside of the VoiceObjects Server framework. For details refer to Declaration above.

The `<collection>` element uses the embedded `<value>` element.

Collection

Attributes

- **`initial`**
  Reference to a Collection, Expression, Script, or Variable object defining the initial value of the collection.

For details on the initial value also refer to Declaration above.

Children

- **`<value>`**
  Optionally defines the constant initial value of the Collection object. It would typically be wrapped in CDATA.
- **`<format>`**
  Defines the output formatting for the Collection object

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<collection name=”Health care plans” initial=”#Build collection”/>


<collection name=”alphabet”>
  <value>
    <![CDATA[
      <root>
        <row>
          <col name=”a”> alpha </col>
          <col name=”b”> beta </col>
        </row>
        <row>
          <col name=”a”> aleph </col>
          <col name=”b”> beth </col>
        </row>
      </root>
    ]]>
  </value>
</collection>
```

Value

Children

- **CDATA**
  Embedded code for a Plug-In or Script object, or embedded value for a Variable object, Collection object, or argument.

Example

```
<value> 42 </value>
```

Working with Collections

Collections are extremely useful when dealing with structured data derived e.g. from a database or other backend system. A variety of operations and expression functions are available to efficiently work with them. More details on these functions can be found in the Expression object entry.

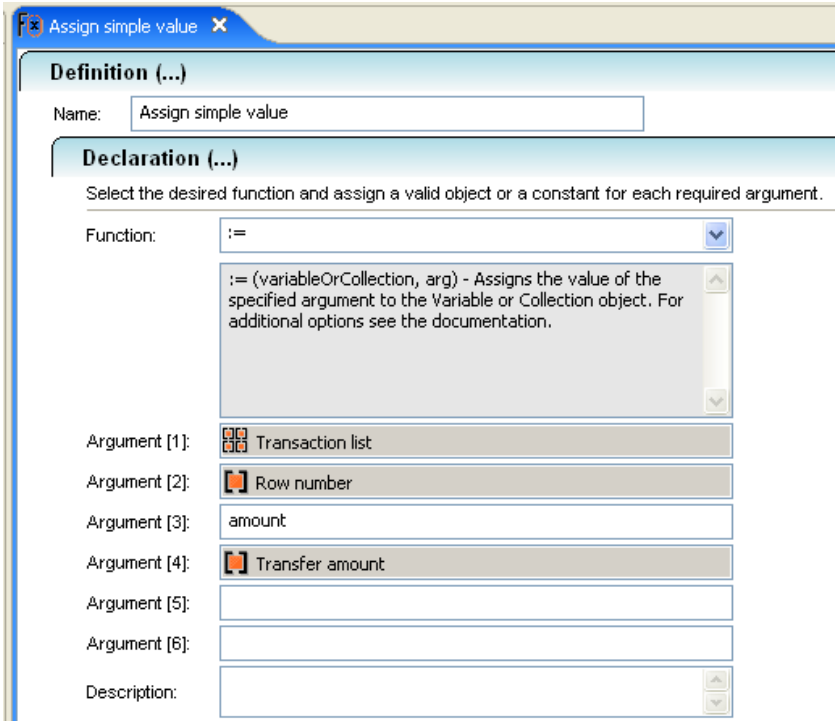
Set and extract values

Individual cell values within a collection can be set and retrieved using the assignment (:=) and the INDEX functions, respectively.

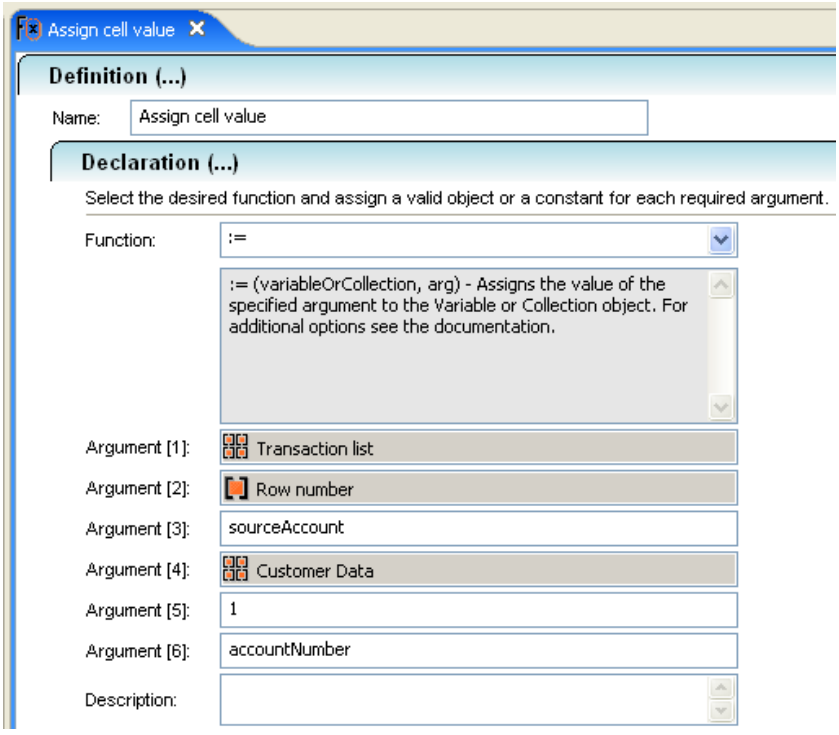
Access to individual cells is facilitated by the INDEX function using row index and column name.

Cells can be assigned simple values such as the content of a variable or a constant by providing the row index and column name as well as the value to be set:

Copyright © 2001-2011 Voxeo Germany GmbH



Alternatively, a cell can be assigned the content of a different cell (from the same or a different collection) by again providing the row index and column name of the cell to be set as well as the collection, row index, and column name from which to retrieve the value:



An entire collection can be emptied using the CLEAR function.

Copyright © 2001-2011 Voxeo Germany GmbH

Find values

Collections often contain lists of certain entries, e.g. recent transactions on a bank account. In such cases it is desirable to be able to find a specific entry in this list based on a given piece of information, e.g. the recipient of a money transfer. Two functions are available for this purpose.

FINDCELL retrieves the value of a specific column given the value in some other column of the same row. It is similar to the SQL statement

```
select field where key=value
```

FINDROW retrieves the index of an entire row given the value in a column of that row. It is similar to the SQL statement

```
select * where key=value
```

Manipulate rows and columns

Sometimes it is desirable to manipulate entire rows or columns within a collection in a single operation. Multiple functions are available for this purpose.

ADDROW and ADDCOLUMN add an entire new row or column to an existing collection. Reversely, DELETEROW and DELETECOLUMN delete an entire row or column from a collection. Finally, REPLACEROW and REPLACECOLUMN replace an existing row or column by an alternate one.

Rows within a collection can be sorted by the values within any of its columns by using the SORTROWS function.

The number of columns and rows in a collection can be retrieved using the COLUMNS and ROWS function, respectively.

The list of column names occurring in a collection is available using the COLUMNNAMES function.

Advanced collection operations

Collections of specific formats (i.e. using specific column names) are also used to provide certain advanced functionalities.

The VALUESUBSTITUTION function facilitates the mapping of certain internal values to multi-lingual external representations. As an example, transaction types represented by internal codes can be mapped to e.g. English, French, and German terms that callers can understand.

The APPLYCONFIGURATION function enables the bulk assignment of values to other Variable, Collection, and Layer objects e.g. to set a caller-specific application configuration. Further details can be found in Application Defaults in Chapter 4 – *Service Deployment* in the *Deployment Guide*.

Usage Examples

Collection objects can be used e.g. to store

- Caller profiles
- Top Ten lists of songs
- Shopping carts

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The following table contains all object types that can reference a Collection object:

Icon	Object Name	Use Case Example
	List	A List object uses Collection objects for both the list data and the selection cart, if defined.
	Expression	A Collection object can be used within Expression objects in a variety of expression functions.
	Connector	A Collection object can be passed as parameters to a Connector object. The Connector object will have access to the collection content as an XML string in the format described in the documentation above.
	Database	A Collection object can be used as the return value in a Database object.
	Script	A Collection object can be passed as parameters to a Script object. The Script object will have access to the collection content as an XML string in the format described in the documentation above.
	Plug-In	A Collection object can be passed as parameters to a Plug-In object. The Plug-In object will have access to the Collection content as an XML string in the format described in the documentation above.
	Log	A Collection object can be used as part of the log statement in a Log object.
	OSDM	A Collection object can be linked within the parameter set of any OSDM object.
Any object that has a Precondition		A Collection object can be used as a Precondition.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Collection object should be as short as possible and at the same time precise enough that other users will know immediately what the Collection object is all about. The description of a Collection object should provide additional information about the structure and form of the content (e.g. number, text, date, time or a Boolean values), the expected or valid value domain and finally something about the initialized value (if available).

The table below is listing various examples as a general guideline:

Copyright © 2001-2011 Voxeo Germany GmbH

Name	Description
Destination Airports	List of valid destination airport names stored in the column *Airport* (e.g. *London Heathrow*).
Customer List	Customer list with two columns containing a 14 digits customer ID and the according name (first and last name).
Traffic News	Actual list of traffic news provided in text form for a particular highway.
Email Inbox	Email inbox with 6 columns: *Sender* (Txt), *Subject* (Txt), *Body* (Txt), *Date* (MM/DD/YYYY), *Importance* (high/low), *Attachment* (Int).
Top 10 Ring Tones	Top down sorted list of the top ten ring tones as complete audio URLs. (e.g. *http://ContentServer/RingTone_4711.wav*).

Copyright © 2001-2011 Voxeo Germany GmbH

Confirmation

Overview

The Confirmation object is a configurable dialog component to confirm and, if necessary, to correct information collected in the preceding dialog. It can be used to confirm individual caller inputs immediately after each is collected, or to confirm a summarized block of items at the end of an input series. If the summary is rejected, the caller can specify which item needs to be corrected.

The Object Definition below covers the configuration of the Confirmation object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Confirmation object belongs to the object category Components.

Dialog Flow Scenario

The first dialog flow example describes a simple confirmation dialog in a flight booking application.

Object – Caller	Dialog Flow
	*You are looking for a flight departing from Boston, which arrives in San Francisco today, June 20th at 9:15 AM. Is this correct?*
	*Yes.*

In the second dialog flow example the caller denies the flight confirmation and the dialog walks the caller through subdialogs to correct the corresponding information.

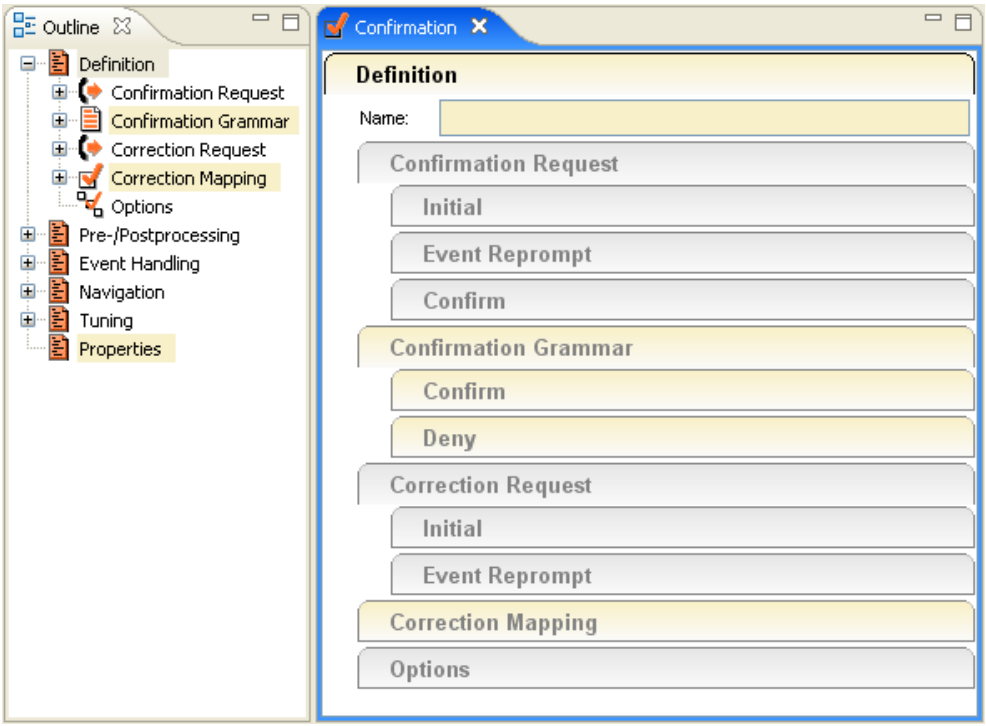
Object – Caller	Dialog Flow
	*You are looking for a flight departing from Boston, which arrives in San Francisco today, June 20th at 9AM. Is this correct?*
	*No.*
	*Which part do you want to change? The departure city, the arrival city, date or time?*
	*The arrival city.*
	*Where do you want to fly?*
	*Denver.*
	*So that's a flight departing from Boston, which arrives in Denver today, June 20th at 9AM. Is this correct?*
	*Yes.*

Copyright © 2001-2011 Voxeo Germany GmbH

Object Definition

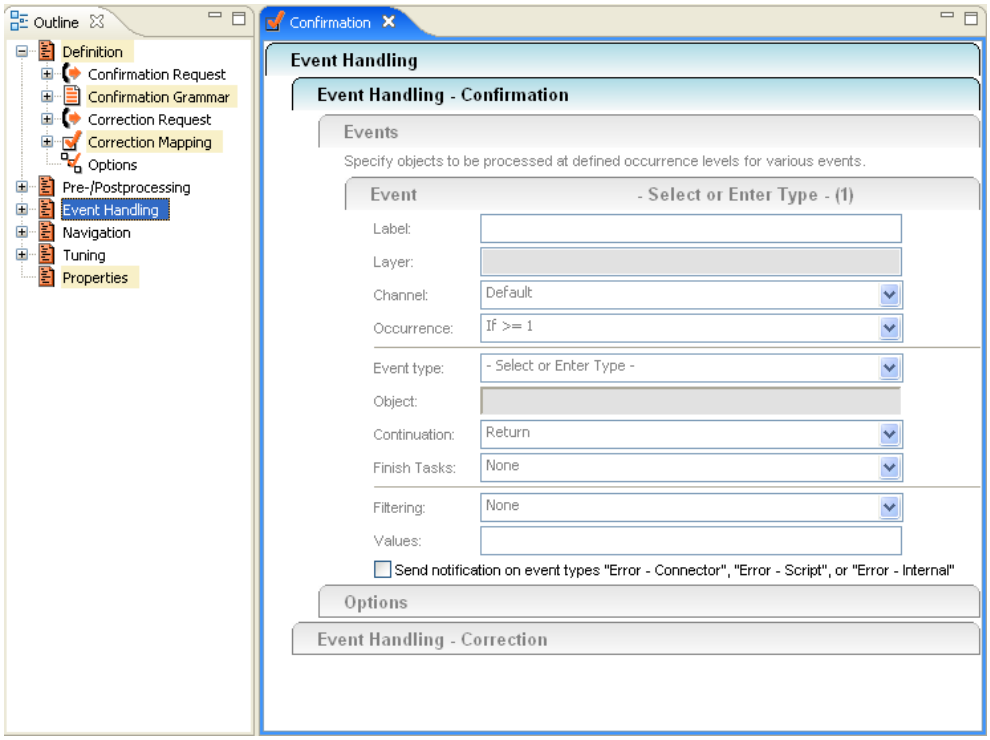
The **Definition** of the Confirmation object provides the following sections:

- Confirmation Request
  To define prompts asking for caller confirmation.
- Confirmation Grammar
  To define acceptable input for confirmation and denial.
- Correction Request
  To define prompts asking for the items that need correction (only in the voice and video channel)
- Correction Mapping
  To specify how items are corrected.
- Options
  To configure processing options for the Confirmation object.



In addition, the **Confirmation** editor provides separate **Event Handling** for the confirmation request as well as for the correction request.

Copyright © 2001-2011 Voxeo Germany GmbH



For further details refer to *Event Handling* in this *Object Reference*.

Also refer to *Pre-/Postprocessing*, *Navigation*, *Tuning*, and *Properties* for details on more additional object configuration.

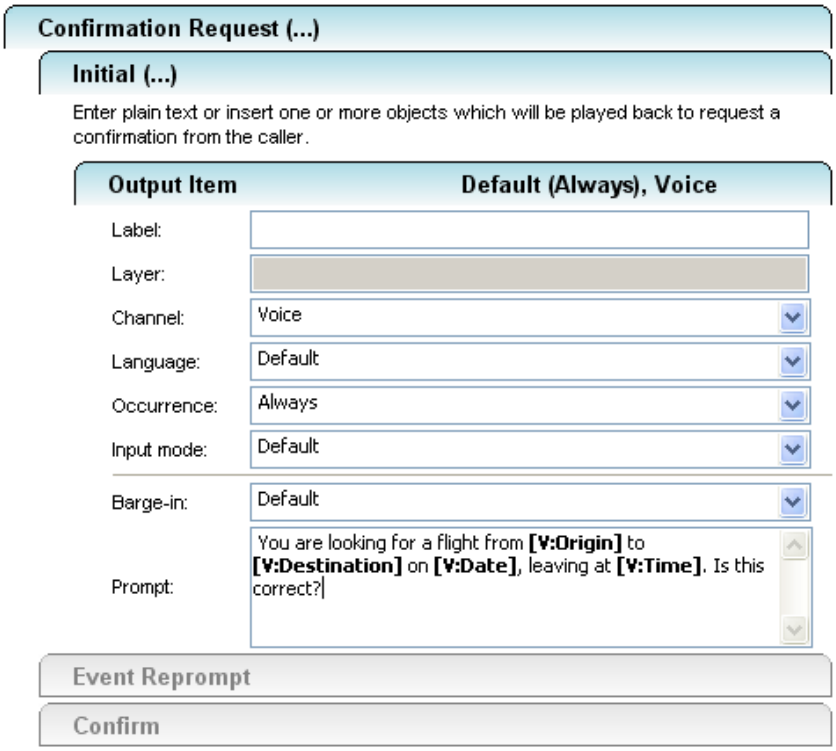
Confirmation Request

The **Confirmation Request** consists of three Output object definitions represented by the **Initial**, **Event Reprompt** and **Confirm** sections. It usually uses a combination of Variable objects, which contain information from preceding subdialogs, and various text elements or prerecorded Audio objects.

The **Initial** section defines the information that needs to be confirmed by the caller. The output defined herein typically summarizes the information gathered so far and asks whether everything is correct.

Copyright © 2001-2011 Voxeo Germany GmbH

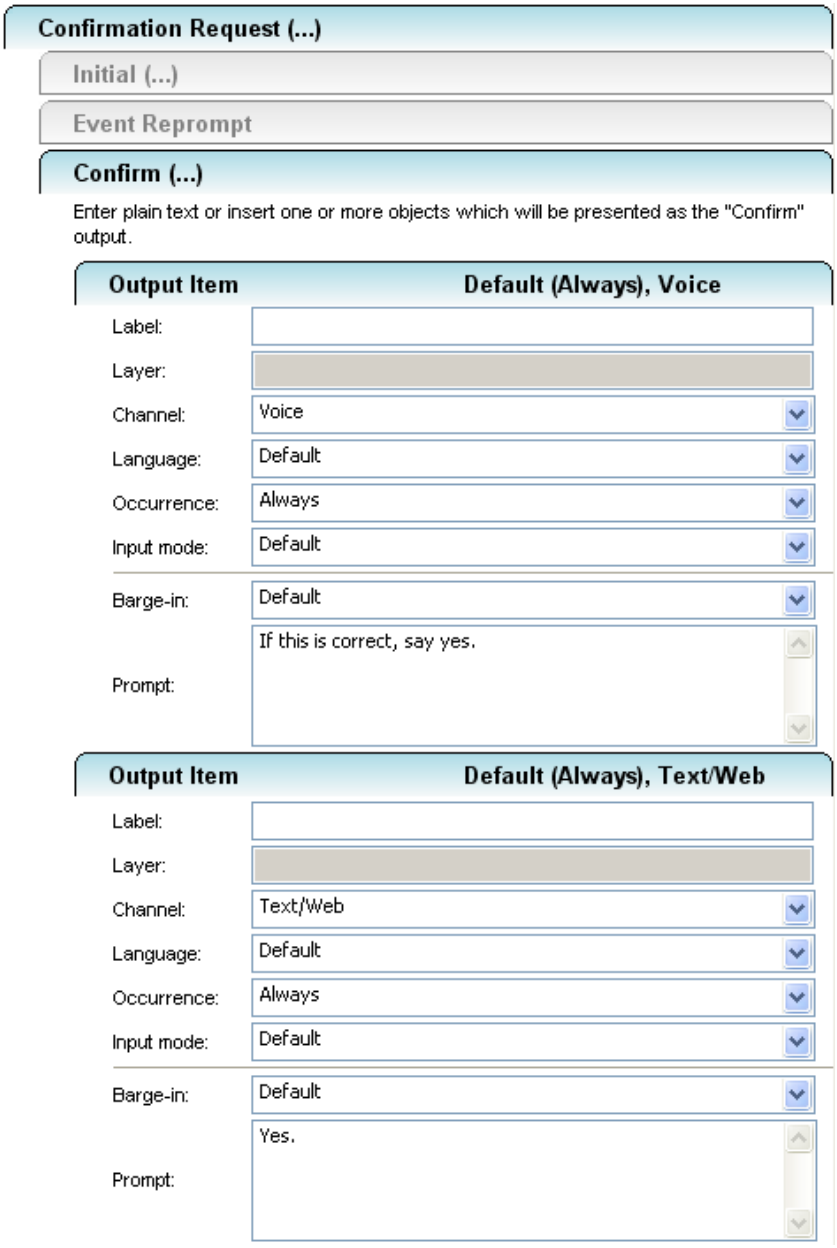
In the example below the Variable objects are displayed using the shortcut notation.



The **Event Reprompt** section defines modified outputs for error cases. This is only processed in the voice and video channel.

The **Confirm** section defines how to present the option to confirm the information. In the voice and video channel, it can be left empty and included in the confirmation request, or it can hold a prompt like *"If this is correct, say yes"*. In the text and Web channel, it must be defined so that the option to confirm can be displayed. In the text channel, the caller can activate it by responding with the access key defined in the **Confirm** grammar (see **Confirmation Grammar**); in the Web channel, the caller activates it through the corresponding hyperlink.

Copyright © 2001-2011 Voxeo Germany GmbH



For further details refer to the Output object in this *Object Reference*.

For a detailed description of how to use the reprompting mechanism, refer to Chapter 8 – *Advanced Event Handling* in the *Design Guide*.

Confirmation Grammar

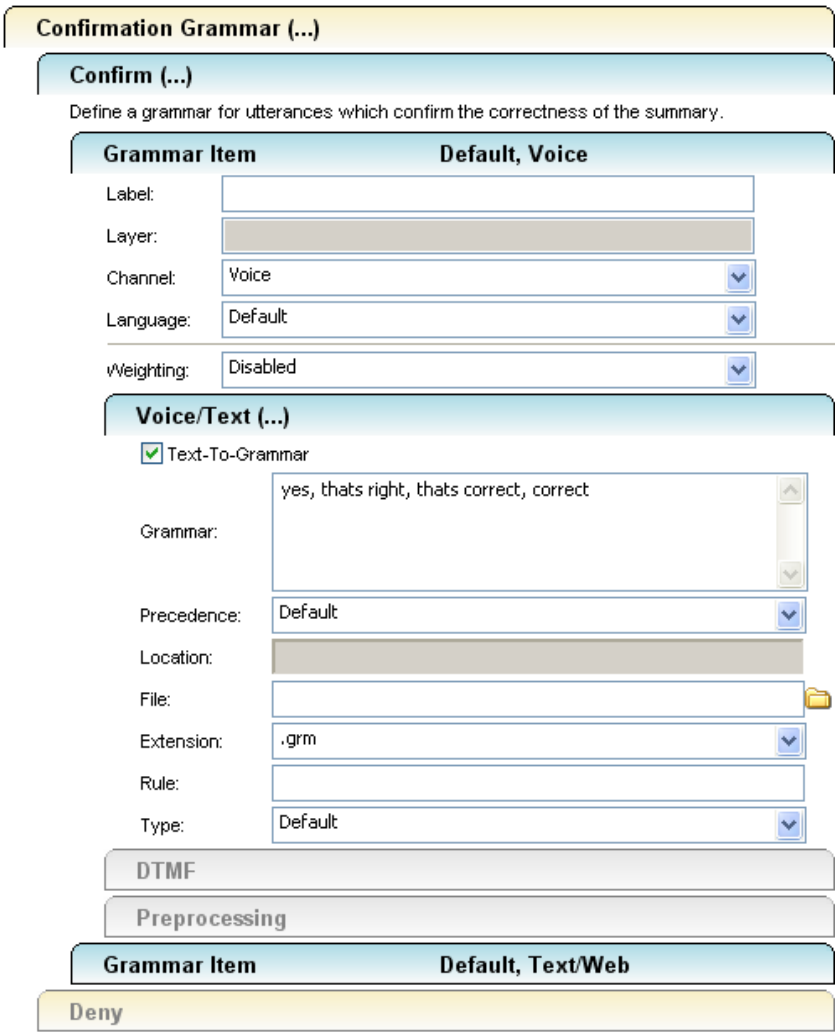
The **Confirmation Grammar** consists of two Grammar object definitions represented by the **Confirm** and **Deny** section.

The **Confirm** section defines the input that confirms the information items presented in the confirmation request. In voice and video applications with speech recognition, this could be a grammar matching utterances like *yes, correct, that's right* etc. In a text application, the option to confirm all information is presented first, i.e. before the correction items, so it should get *1* as the grammar.

Copyright © 2001-2011 Voxeo Germany GmbH

The **Deny** section defines the input that indicates that at least one of the information items is wrong. The deny grammar is only processed in the voice and video channel; in the text and Web channel, there will not be an explicit option to deny, instead the correction items (i.e. how to correct each individual piece of summarized information) will be presented immediately.

Neither of these two grammars needs to fill any slots.



For further details refer to the Grammar object in this *Object Reference*.

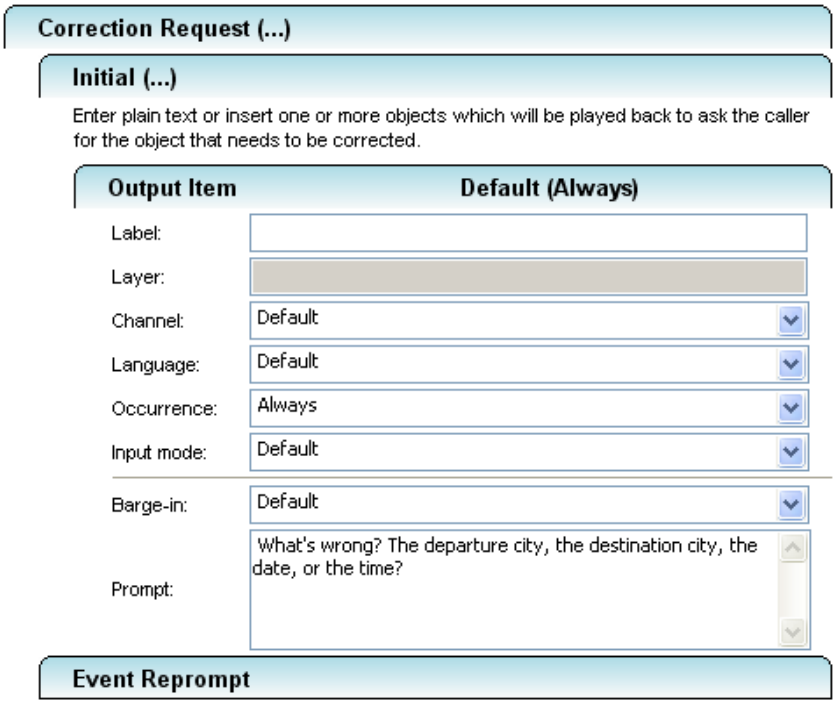
Correction Request

The **Correction Request** section is used to ask the caller which information item needs to be corrected, in case the confirmation request was denied. Usually plain text or Audio objects are used here.

If the Confirmation object has only one correction item, the correction request is played and the object of the correction item is processed right after it. If more than one correction item is defined, the correction request is an input state, so that the application waits for the caller to say which item was wrong. It has its own event handling section which is then applied.

Copyright © 2001-2011 Voxeo Germany GmbH

The correction request is only processed in the voice and video channel, as the option to deny the confirmation request is not offered in the text and Web channel.

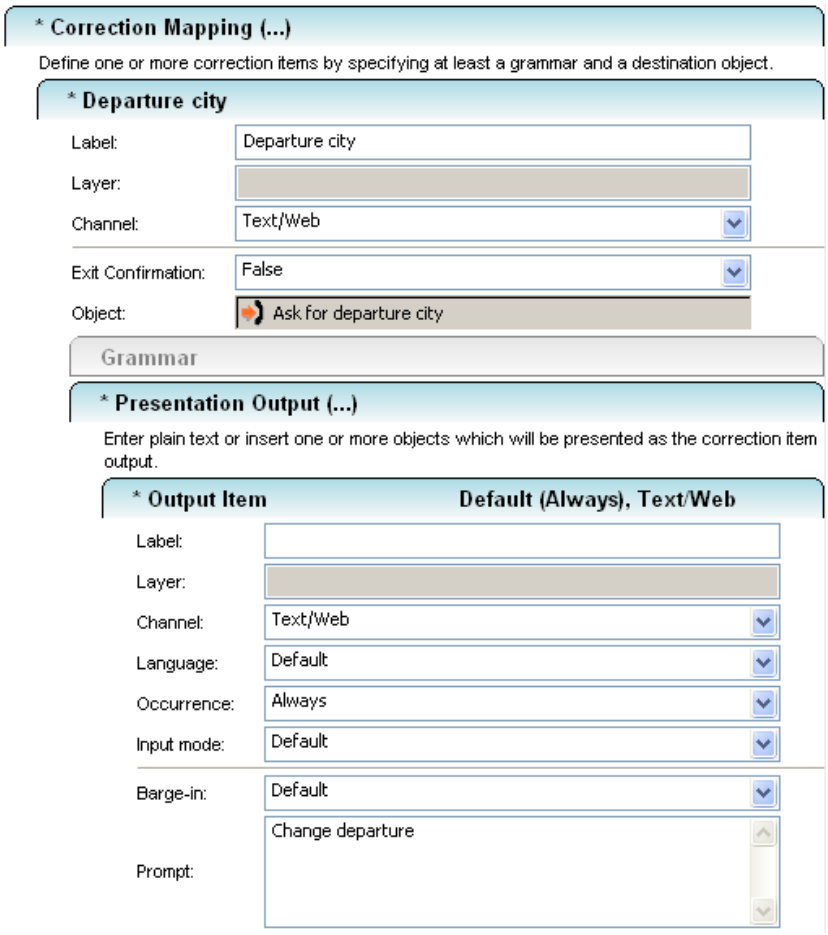


For further details refer to the Output object in this *Object Reference*.

Correction Mapping

The **Correction Mapping** consists of one or more correction items.

Copyright © 2001-2011 Voxeo Germany GmbH



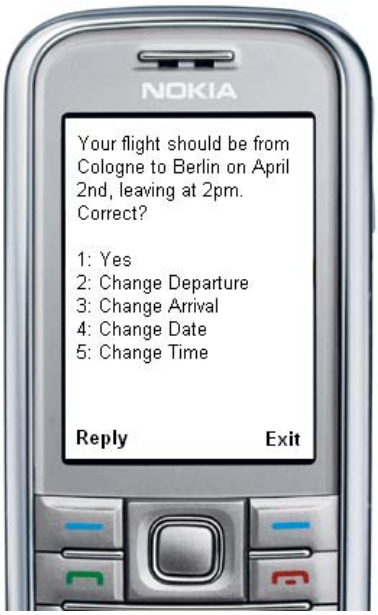
In each correction item the following properties can be specified:

Property	Description
Label	Optional parameter to identify the correction item in a list.
Layer	Optional parameter to indicate the layer that the correction item belongs to. For more information on layers, see *Layers* in this *Object Reference* or Chapter 7 – *How to Use Layers* in the *Design Guide*.
Channel	Identifies the channels in which the respective correction item is to be active. If a correction item is appropriate for all channels, the default channel layer setting *Default* should be used. Otherwise, define one of the four channels *Voice*, *Video*, *Text* or *Web*. For your convenience, use the two combined settings *Voice/Video* or *Text/Web* in case the correction item is to be active in those combinations. Using a combined channel is the same as defining two correction items with the individual channel setting, with all other settings being the same.
Exit Confirmation	Indicates what to do after the object defined for this correction item has been processed. The default setting *False* indicates that the dialog returns to the Confirmation object, so that after correcting an item the

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
	summary is presented again through the confirmation request. When set to *True*, the processing of the Confirmation object is terminated and the dialog continues with the next object after the Confirmation object (if any). The **Exit Confirmation** property can dynamically be set at call time by defining it with a Variable, Expression, or Script object. If it is dynamically set, the value is determined before the object defined for the correction item is processed.
Object	A mandatory object reference that points to the object that processes the correction, typically an Input or a Sequence object, asking again for the respective information.
Grammar	Specifies the caller input that identifies this particular item as the one that needs to be corrected (if more than one correction item is defined). The input is a response to the correction request. In voice and video applications with speech recognition, this could be a grammar matching utterances like *the date, the date is wrong* etc. In a text application, the correction items are presented after the confirm grammar and in the order in which they are defined in the object (see Presentation Output), so they should get consecutive keys, typically starting with *2* (since 1 is used to confirm all information). If you want to let the USSD browser do auto-numbering on the resulting items, define question mark "?" as the grammar.
Presentation Output	Defines how to present the correction item to the caller. It will be played after the confirmation request, in the order in which the items are defined in the object. In the voice and video channel, the presentation output can be left empty if you do not wish a special prompt to explain how to correct an item. Otherwise it could be something like "*To correct the departure city, say departure*". In the text and Web channel, it must be defined to display the correction item; otherwise the caller has no way of activating it. For the example of correcting the date it could be "*Change Departure*".

Copyright © 2001-2011 Voxeo Germany GmbH



The processing of the destination object usually results in Variable objects being filled with different values (e.g. different *departure cities*). After the destination object has been processed, the Confirmation object starts again from the top, i.e. with the confirmation request.

Tip: By providing the appropriate grammars and destinations, the caller can correct both individual items such as an *arrival city* as well as combinations of items such as *date* and *time*. However there is no automatic merging of grammars and destination objects, so individual correction items need to be provided for each combination. If you want the caller to be able to say, e.g., "*the date and time are wrong*" in a voice or video application with speech recognition, define a separate correction item with a corresponding grammar and a Sequence object that processes two Input objects, the first one asking for the date, the second one asking for the time again.

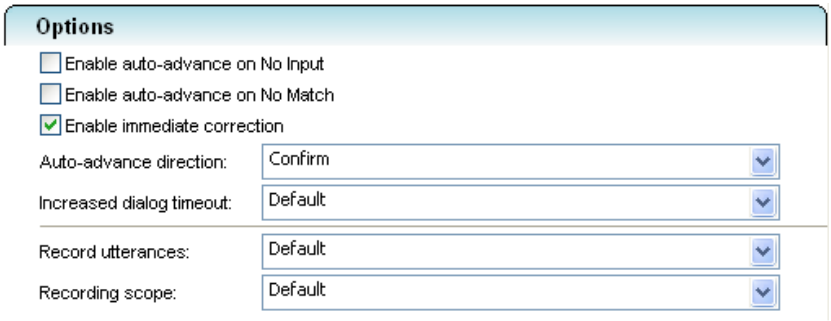
Note: During all inputs within the Confirmation object in the voice and video channel, the confirm grammar, the deny grammar, and the grammars of all correction items are active. Depending on how these grammars are built, this enables the designer to create Confirmation objects in which the caller can immediately say (in response to the confirmation request) which items need to be corrected, instead of having to wait for the second question asked in the correction request.

In the special case of a *single* correction item in the voice and video channel, it is not required to define a grammar within this item since the item is activated if the caller input matches the deny grammar defined in the **Confirmation Grammar** section. Note that the correction request is still played if defined. This is to enable outputs such as "*OK, let's try this again*" before processing the object that allows the correction (typically an Input or a Sequence object).

Copyright © 2001-2011 Voxeo Germany GmbH

Options

The Confirmation object provides the setting of six options in the **Options** section.



The option **Enable auto-advance on No Input** enables auto-advance in case the caller does not respond to the confirmation request (only supported in the voice and video channel). This is useful to allow “silent” confirmation, if desired. If the check box is selected, a No Input event finishes the processing of the Confirmation object and proceeds to the next object in the dialog flow instead of processing the event handling specifications. Thus, not saying anything has the same effect as saying something that is matched by the confirm or deny grammar (depending on the setting of the **Auto-advance direction** option). By default, the check box is clear.

The option **Enable auto-advance on No Match** enables auto-advance in case the caller says something in response to the confirmation request that does not match the grammar (only supported in the voice and video channel). If the check box is selected, a No Match event finishes the processing of the Confirmation object and proceeds to the next object in the dialog flow instead of processing the event handling specifications. Thus, saying something that does not match the active grammars has the same effect as saying something that is matched by the confirm or deny grammar (depending on the setting of the **Auto-advance direction** option). By default, the check box is clear.

The option **Enable immediate correction** controls whether callers can immediately use an utterance from one of the correction item grammars while in the initial confirmation question (the default). Sometimes, e.g. when working with pure DTMF grammars, this is not desirable and can then be suppressed.
Note that this option is only relevant for the voice and video channel. In the text and Web channel, there is no explicit confirmation phase.
By default, the check box is selected.

The **Auto-advance direction** option determines in which direction the dialog flow proceeds in case auto-advance has been enabled and a No Input or No Match event occurs. If it is set to *Confirm*, the dialog proceeds with the object coming right after the Confirmation object. If it is set to *Deny*, the dialog proceeds with the correction request.

The option **Increased Dialog Timeout** specifies a timeout value that is applied for this object in case it is higher than the current dialog timeout in a session. If it is lower, the setting is ignored. It is mainly applicable in the text and Web channel, where the dialog timeout of a service should be set to a relatively low value, e.g. 40 seconds, to keep the number of concurrent sessions low, as they usually run into the server timeout due to technical reasons. Use this setting to increase the timeout temporarily for objects that are likely to require more time to respond to, e.g. when the caller has to type in an address.

Copyright © 2001-2011 Voxeo Germany GmbH

The option **Record utterances** specifies whether utterances made in this object are to be recorded (*True*) or not (*False*), given that utterance recording is switched on for this session. *Always* indicates that utterances in this object are always recorded when utterance recording is enabled on the service, regardless of statistical filtering. By default the value is *Default*, indicating that the current value of the corresponding dialog context setting RECORDUTTERANCES is used.

The option **Recording scope** defines the scope in which recordings will be made. If set to *Recognition*, only those utterances will be recorded that were successfully recognized (including the activation of hyperlinks). If set to *No Match*, only those utterances will be recorded that triggered a No Match event. If set to *All*, any recording will be stored. If set to *Default*, the current value of the corresponding dialog context setting RECORDINGSCOPE is used. By default the value is *Default*.

Due to the nature of recordings, the feature of recording utterances is only available in the voice and video channel. For more information on this refer to Chapter 6 – *Recording of Utterances* in the *Deployment Guide*.

Usage Examples

The following two examples demonstrate the use of a Confirmation object in a voice application to confirm a single caller input and several inputs at once, respectively.

Confirm a single input

This example shows how to define a Confirmation object to confirm a single caller input. The **Correction Mapping** section holds only a single item with no grammar definition. Using this Confirmation object, the caller can confirm a PIN entered earlier in the dialog.

1. Open an empty **Confirmation** editor.
2. Enter *Confirm PIN* as the name of the Confirmation object.
3. In the **Confirmation Request** section, enter the initial prompt of the confirmation request, like *You said [V:PIN]. Is that correct?*, assuming the caller input is stored in a Variable object *PIN*.
4. In the **Confirm** section of the **Confirmation Grammar**, enter a grammar representing confirming utterances (like *yes*, *okay*, or *correct*, etc.). Then enter a grammar representing rejecting utterances (like *no*, *not correct*, etc.) in the **Deny** section.
5. In the **Correction Request** section, enter the initial prompt of the correction request, like *Sorry, let's try it again*. This prompt is played if the caller denies the confirmation request.
6. In the **Correction Mapping** section, define a single correction item by linking the Input object that asks the caller to enter the PIN. It can be the same one that was initially used, before the Confirmation object. Leave its **Grammar** section empty, as this correction item is immediately processed on denial of the confirmation request.

When the caller confirms the repeated PIN, the dialog proceeds with the following object in the dialog flow. When the caller denies the recognized PIN, the Input object asking again for the PIN is processed. After this, the server returns to the Confirmation object, starting all over with the confirmation request, until the caller finally confirms.

Copyright © 2001-2011 Voxeo Germany GmbH

Confirm multiple inputs

This example shows how to define a Confirmation object to confirm several caller inputs in one dialog step. The **Correction Mapping** section holds two items, to allow the correction of the account number or PIN that the caller entered.

1. Open an empty **Confirmation** editor.
2. Enter *Confirm Account Number and PIN* as the name of the Confirmation object.
3. In the **Confirmation Request** section, enter the initial prompt of the confirmation request, like *So your account number is [V:Number] and your PIN is [V:PIN]. Is that correct?*, assuming the account number is stored in a Variable object *Number* and the PIN is stored in a Variable object *PIN*.
4. In the **Confirm** section of the **Confirmation Grammar**, enter a grammar representing confirming utterances (like *yes*, *okay*, or *correct*, etc.). In the **Deny** section, enter a grammar representing rejecting utterances (like *no*, *not correct*, etc.).
5. In the **Correction Request** section, enter the initial prompt of the correction request, like *I'm sorry. What's wrong, the account number or the PIN?*. This prompt is played if the caller denies the confirmation request.
6. In the **Correction Mapping** section, define the first correction item by linking the Input object that asks the caller to provide the account number. It can be the same one that was initially used, before the Confirmation object. Enter a grammar for this correction item representing utterances that select *account number* as the item to correct, like *account number*, *the account number*, or *the number is wrong*, etc.
7. Define the second correction item by linking the Input object that asks the caller to provide the PIN. It can be the same one that was initially used, before the Confirmation object. Enter a grammar for this correction item representing utterances that select *PIN* as the item to correct, like *PIN*, *the PIN*, or *the PIN is wrong*, etc.
8. Define a third correction item that allows the caller to correct both items in case the speech recognizer got both utterances wrong. Link a Sequence object that holds both Input objects from steps 6 and 7. Enter a grammar representing utterances that indicate both items are wrong, like *both*, *both are wrong*, *neither is correct*, etc.

In this example, the caller can correct either the account number or the PIN. In case both items are wrong, the caller can say *both*, in which case he or she is asked for both the number and the PIN again.

VoiceObjectsXML Definition

The Confirmation object is represented by the VoiceObjectsXML element **<**`confirmation`**>**. It has six attributes and fifteen groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The **<**`confirmation`**>** element uses the embedded **<**`correctionItem`**>** element.

Confirmation

Attributes

- **`autoAdvanceNoInput`**
  Define whether the dialog should automatically advance in case of a No Input event. Either *true* or *false*. If not specified, defaults to *false*.

Copyright © 2001-2011 Voxeo Germany GmbH

- **autoAdvanceNoMatch**
  Define whether the dialog should automatically advance in case of a No Match event. Either *true* or *false*. If not specified, defaults to *false*.
- **autoAdvanceDirection**
  Defines the direction of an auto advance. Can be either *confirm* or *deny*. If not specified, defaults to *confirm*.
- **recordUtterances**
  Defines whether to use utterance recording. One of *true*, *false*, *always*, *default*. If not specified, defaults to *default*.
- **recordingScope**
  Defines the scope for utterance recording. One of *all*, *noMatch*, *recognition*, *default*. If not specified, defaults to *default*.
- **immediateCorrection**
  Defines whether utterances from the correction item grammars are allowed in the confirmation step. Either *true* or *false*. If not specified, defaults to *true*.

Children

- **<expression usage="precondition">** or
  <variable usage="precondition"> or
  <collection usage="precondition"> or
  <script usage="precondition">
  Defines the precondition for the Confirmation object.
- **<sequence usage="preprocessing">**
  Defines the preprocessing sequence for the Confirmation object.
- **<output usage="confirmation" type="initial">**
  Defines the "initial" section of the confirmation request.
- **<output usage="confirmation" type="reprompt">**
  Defines the "event reprompt" section of the confirmation request.
- **<output usage="confirmPresentation">**
  Defines the confirm presentation output.
- **<grammar usage="confirm">** [required]
  Defines the grammar used to confirm the summary.
- **<grammar usage="deny">** [required]
  Defines the grammar used to deny the summary.
- **<output usage="correction" type="initial">**
  Defines the "initial" section of the correction request.
- **<output usage="correction" type="reprompt">**
  Defines the "event reprompt" section of the correction request.
- **+<correctionItem>**
  Defines the list of correction items.
- **<eventHandling usage="confirmation">**
  Defines the event handling for the confirmation phase of the Confirmation object.
- **<eventHandling usage="correction">**
  Defines the event handling for the correction phase of the Confirmation object.
- **<customNavigation>**
  Defines the navigation settings for the Confirmation object.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<tuning>**
  Defines the tuning settings for the Confirmation object.
- **<sequence usage=”postprocessing”>**
  Defines the postprocessing sequence for the Confirmation object.

Example

```
<confirmation>
  <output usage="confirmation">
    <outputItem>
      <text>You want the insurance premium for a </text>
      <variable link=”#Car manufacturer”/>
      <text> </text>
      <variable link=”#Car model”/>
      <text>, right?</text>
    </outputItem>
  </output>
  <grammar usage="confirm">
   <grammarItem>
      <grammarDefinition mode="voice">
        yes, yeah, right, correct
      </grammarDefinition>
    </grammarItem>
  </grammar>
  <grammar usage="deny">
    <grammarItem>
      <grammarDefinition mode="voice">no</grammarDefinition>
    </grammarItem>
  </grammar>
  <output usage="correction">
    <outputItem>
      <text>What's wrong, the manufacturer or the model?</text>
    </outputItem>
  </output>
  <correctionItem>
    <grammar>
      <grammarItem>
        <grammarDefinition mode="voice">
         manufacturer
        </grammarDefinition>
      </grammarItem>
    </grammar>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
    <input link="#Enter car manufacturer"/>
  </correctionItem>
  <correctionItem>
    <grammar>
      <grammarItem>
        <grammarDefinition mode="voice">
          model
        </grammarDefinition>
      </grammarItem>
    </grammar>
    <input link="#Enter car model"/>
  </correctionItem>
</confirmation>
```

CorrectionItem

Attributes

- **`label`**
  A text string providing a name for the correction item.
- **`layer`**
  Defines the layer for the correction item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **`channel`**
  Defines the channel(s) for which this correction item is valid. Can be *default*, *voice*, *video*, *text*, *web*, *voiceVideo*, or *textWeb*. If not specified, defaults to *default*.
- **`exit`**
  Defines whether the dialog exits the Confirmation object after processing this correction item. Either *true* or *false*. If not specified, defaults to *false*.

Children

- **`<grammar>`**
  Defines the activation grammar for the correction item. Optional if only a single correction item exists; otherwise it is required.
- **`<output usage=”presentation”>`**
  Defines the optional presentation for the correction item.
- One of the following:
  `<module>`, **`<input>`**, **`<output>`**, **`<sequence>`**, **`<menu>`**, **`<confirmation>`**, **`<list>`**, **`<osdm>`**, **`<plugin>`**, **`<connector>`**, **`<script>`**, **`<log>`**, **`<if>`**, **`<case>`**, **`<goto>`**, **`<loop>`**, **`<expression>`**, **`<recording>`**, **`<transfer>`**, **`<pause>`**, **`<exit>`**
  Defines the object to be processed if the correction item is activated.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<correctionItem>
  <grammar link="#Correct model grammar"/>
  <input link="#Enter car model"/>
</correctionItem>
```

Object Interoperability

The following table contains all object types that can reference a Confirmation object:

Icon	Object Name	Use Case Example
	Module	A Confirmation object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, a Confirmation object is used within a Sequence object in a dialog flow.
	Menu	A Confirmation object can be used as the destination object within a Menu item of a Menu object.
	Confirmation	A Confirmation object can be used as the destination object within a correction item inside another Confirmation object.
	If	A Confirmation object can be used in either the THEN or the ELSE item of an If object.
	Case	A Confirmation object can be used in any WHEN item of a Case object.
	Loop	A Confirmation object can be used within the embedded sequence of a Loop object.
	Goto	A Confirmation object can be referenced via a Goto object.
	Hyperlink	A Confirmation object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Confirmation object should identify which data items are being confirmed. The short description should contain a brief explanation of the capabilities and restrictions (e.g. only one item can be corrected vs. multiple items can be corrected at once). The table below lists three examples:

Name	Description
 Confirm flight booking	Presents the entire flight booking, including origin, destination, date, and time. Allows the correction of each individual item.

Copyright © 2001-2011 Voxeo Germany GmbH

Name	Description
Confirm date and time	Presents date and time. Either one or both can be corrected.
Confirm transaction	Asks caller whether the transaction should be completed as presented. If not, the entire input process for the transaction is repeated.

Copyright © 2001-2011 Voxeo Germany GmbH

Connector

Overview

The Connector object enables access to external data sources and back-end systems. It provides an open API (Application Programming Interface) for bi-directional data exchange, such that dynamic content can be integrated into the dialog flow. The Connector object can also be used to establish a connection with various other communication channels such as SMS, MMS, e-mail, fax or Web sites.

The Connector object supports three different ways to execute operations against external systems: CGI scripts (Common Gateway Interface - i.e. JSP, ASP, Perl script, etc.), Java programs, and Web services calls.

All of them support the exchange of data through an optional parameter set.
For CGI scripts, parameters are part of the HTTP request and the script needs to return to the connector a result set in a specific XML format that contains the result of the requested operation (e.g. the outcome of a database query).
When using Java code, a simplified bean interface is used to exchange data with getter and setter methods.
Web Service connectors can receive a set of parameters and typically return an XML structure that will be made available to the application through a Variable or Collection object.

Usage examples for the Connector object are:

- Retrieving dynamic content from a content management system
- Accessing information through Web services
- Retrieving customer profiles from CRM (Customer Relationship Management) systems at the beginning of the call and writing back updates after the call has ended
- Making a transaction on a back-end system, e.g. booking a flight
- Sending an e-mail or a text message (SMS) to confirm a transaction

HTTP Connectors can use an HTTP proxy, which can be set in a Service object. For more information on this, refer to Default parameters in Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*.

The Object Definition below covers the configuration of the Connector object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Connector object belongs to the object category Resources.

Dialog Flow Scenario

The following dialog flow shows a database transaction while the caller is logging into a brokerage service. The personal stock portfolio of the caller is retrieved and played back as a list.

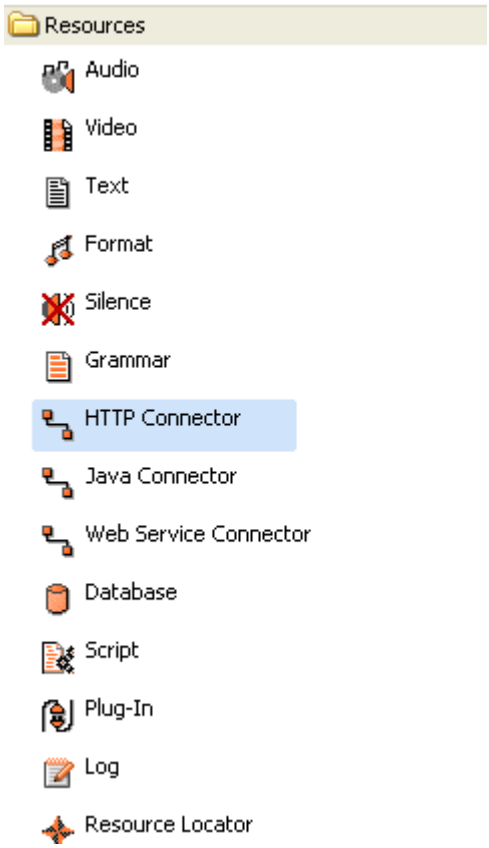
Object – Caller		Dialog Flow
		Please wait while I check your data.

Copyright © 2001-2011 Voxeo Germany GmbH

Object – Caller		Dialog Flow
		A wait loop audio file is played until the connector has finished.
		Connects to a Java program that accesses a database holding customer data.
		[*Earcon for positive confirmation*]
		Welcome to the brokerage service. Here's a list of all the companies in your portfolio…

Object Definition

In Desktop for Eclipse, the different Connector types are represented by three different Connector objects, HTTP Connector, Java Connector and Web Service Connector which may be selected from the **Object Palette**.



The configuration of each Connector object type is described in the following sections.

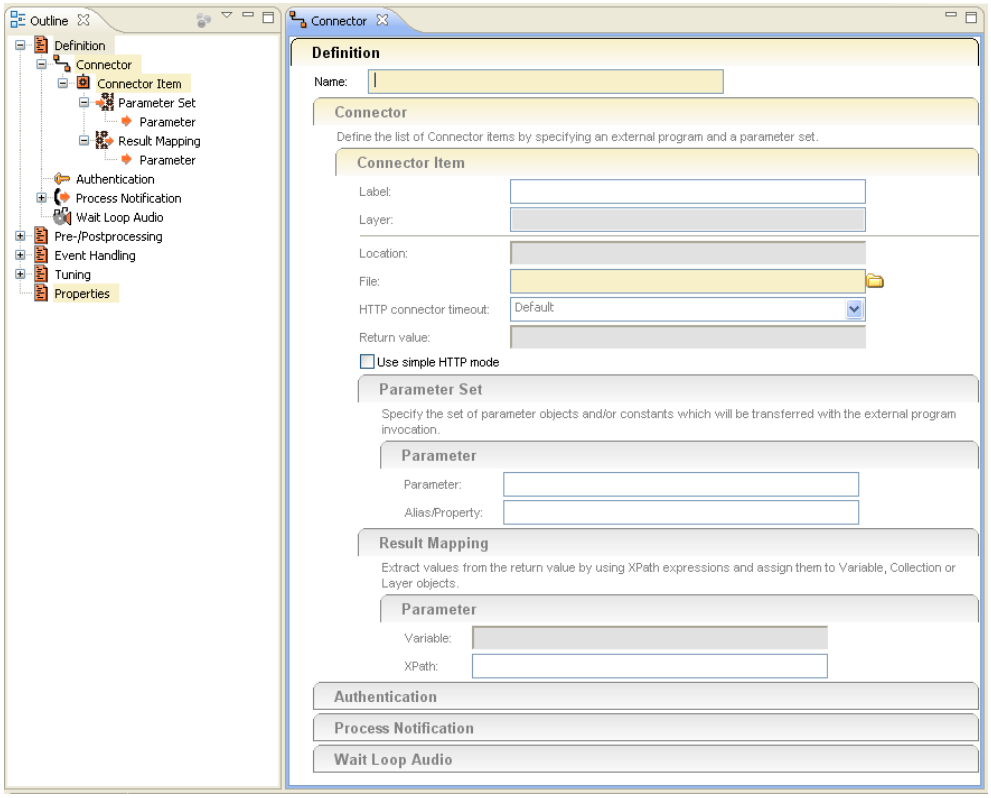
HTTP Connector Definition

The **Definition** of a HTTP Connector object provides the following sections:

- Connector
  To define one or more Connector items.

Copyright © 2001-2011 Voxeo Germany GmbH

- Authentication
  To define the user ID and password for HTTP authentication, and a certificate URL for HTTPS connections.
- Process Notification
  To optionally specify a notification to the caller that the connector processing has started.
- Wait Loop Audio
  To optionally specify an audio file that is played back while the connector process is running.

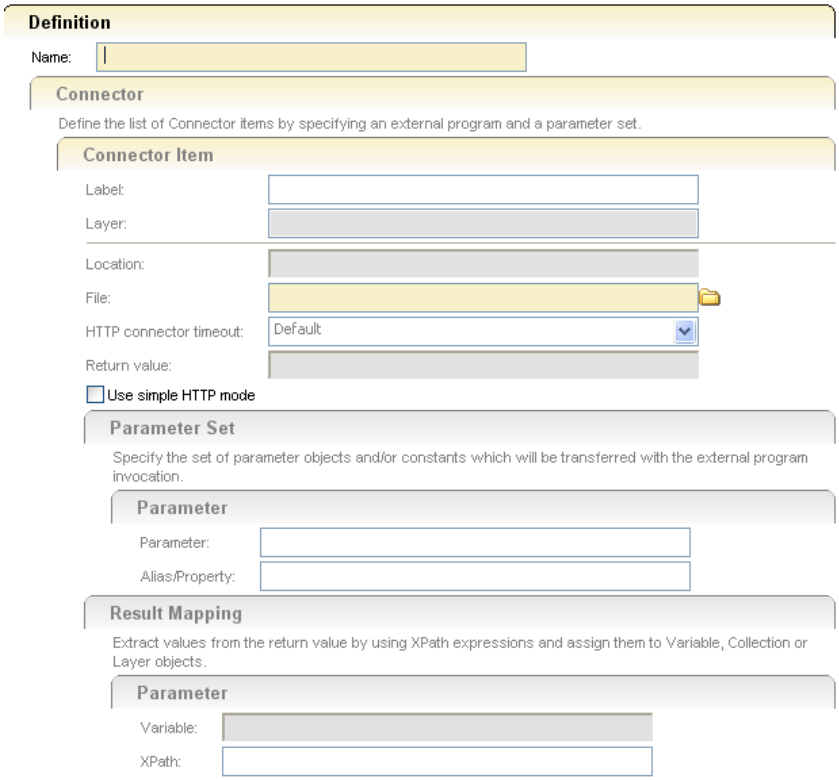


For further details regarding additional object configuration, refer to *Pre-/Postprocessing*, *Event Handling*, *Tuning*, or *Properties* in this *Object Reference*.

Connector

In the **Connector** section, define at least one **Connector Item** that will process a CGI script.

Copyright © 2001-2011 Voxeo Germany GmbH



Each **Connector Item** contains a set of fields:

Property	Description
Label	Optional parameter to identify the Connector item in a list.
Backend ID	Optional parameter to identify the back-end component that is accessed by the Connector item.
Layer	Optional parameter to indicate the layer that the Connector item belongs to. For more information on layers, see Chapter 7 – *How to Use Layers* in the *Design Guide* or the Layer object in this *Object Reference*.
Location	Defines the URL where the CGI script is located.
File	Defines the name of the CGI script file.
HTTP connector timeout	Specifies the time after which the request is aborted if no answer is received from the back-end. If the connector takes longer to process than the time specified in this field, an *Error – Connector* event is activated. You can select one of the values of the drop-down list or enter a custom value. Custom values are interpreted as seconds. If you leave it at *Default*, the setting from the corresponding Service object definition is taken. See Processing the Connector Object for interactions with the VoiceXML *fetchtimeout* property.
Return value	Allows to link a Variable, Collection, or Layer object which will be set to the return value of the connector. The return value is used in two different ways: In the default mode, the return value is the HTTP return code

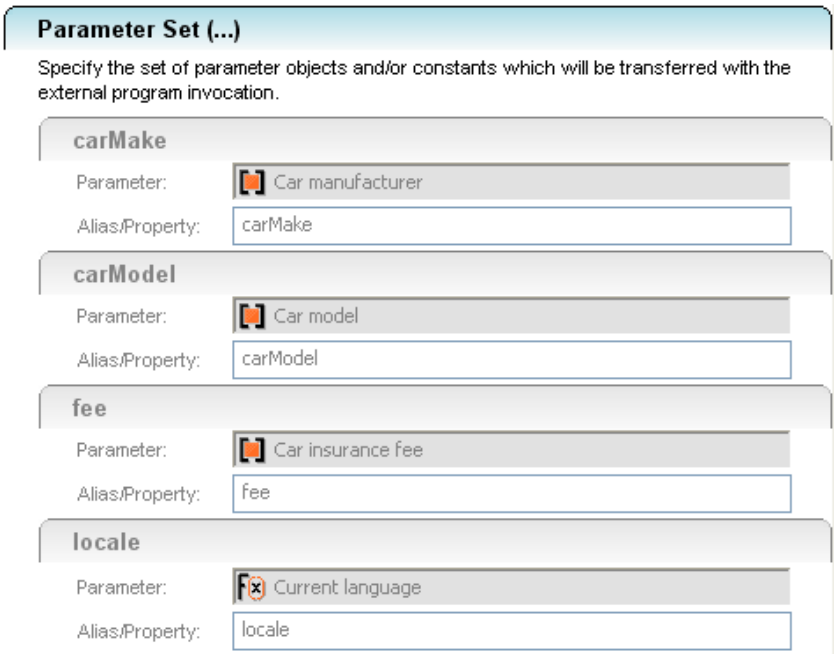
Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
	received from the Web application server that took the request. It can be 200 for successful requests, 404 for "File not found" exceptions, or any other HTTP return code. Note that the return value is set even if the back-end access did not succeed, e.g. in case of a 404 error. If the connector fails because it runs into the HTTP connector timeout (see above), the return value is set to 999. When the option **Use Simple HTTP Mode** is selected, the return value receives the actual return value produced by the CGI script.
Use simple HTTP mode	Indicates that instead of using the XML data format described in Exchanging Data via an XML API, the parameters defined in the **Parameter Set** are sent as simple HTTP POST parameters. In this case the single return value of the CGI script is assigned to the **Return Value** instead of the HTTP response code.

The individual Connector items are processed one after the other in the order in which they are defined. The failure of any one of these can be handled by defining event handling for the event type *Error – Connector*. For more information, refer to *Event Handling* in this *Object Reference*. If no such event handling is defined and a Connector item fails, an error message is played and the dialog ends.

Parameter Set

The **Parameter Set** section of a **Connector Item** enables the exchange of processing information with the CGI script. Possible parameter types are constants, Resource Locator, Variable, Collection, Expression, Layer, and Script objects. The parameter set is optional.



Any number of parameters can be exchanged with a connector.

Copyright © 2001-2011 Voxeo Germany GmbH

The **Alias/Property** field is optional (except for constants) and can be used to assign a name to the parameter by which it can be referenced within the script. If the field is left empty, the reference ID of the respective object is used. For constants, providing an alias is required (otherwise they will not be passed to the script).

When working in **Request Mode** *REST*, it is possible to define the entire request body by setting the **Alias/Property** field to POST (when using **Method** Post) or PUT (when using **Method** Put).

HTTP request header parameters can be defined by using the notation *#HeaderName#* in the **Alias/Property** field. So e.g. to set the “Accept” header, define a parameter with **Alias/Property** *#Accept#* and the list of accepted media types provided as **Parameter**.

The paragraph Exchanging Data via an XML API explains how the CGI script can pass back data.

Result Mapping

The **Result Mapping** section allows extracting specific values applying an XPath expression to the return value. Any number of result mappings can be defined. The expression specified in the **XPath** field will be applied to the return value of the connector. The result of this operation will be assigned to the Variable, Collection or Layer object defined in the **Variable** field. As XPath is designed to retrieve values from a well-formed XML structure, the Connector item must return such a value. If the Connector item returns nothing or a non-XML value the *Error – Internal* event is activated.

An exception to this rule is if the Connector item returns a valid JSON structure – this is automatically converted into XML and can then be accessed in the same way. The conversion is the same as that used by the JSON2XML Expression function.

The **Return Value** field must not necessarily contain a Variable, Collection or Layer object. The result mapping items will be assigned nevertheless. If this field contains an object, the unchanged return value will be assigned to it.

Example:

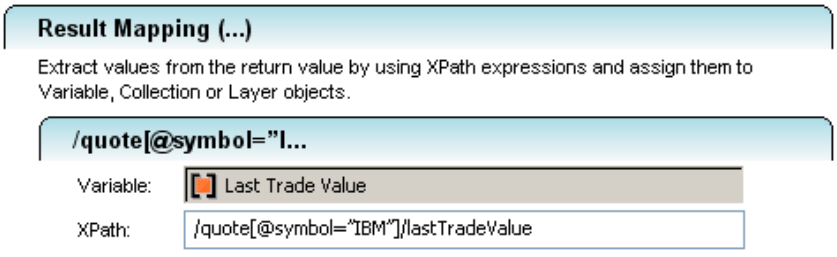
Given a Connector item that returns the following XML structure:

```
<quote symbol=”IBM”>
    <lastTradeValue>82.17</lastTradeValue>
    <previousCloseValue>82.16</previousCloseValue>
</quote>
```

The following XPath expression can be used to retrieve the last trade value:

```
/quote[@symbol=”IBM”]/lastTradeValue
```

The **Result Mapping** section in the **Connector** editor would look like this:

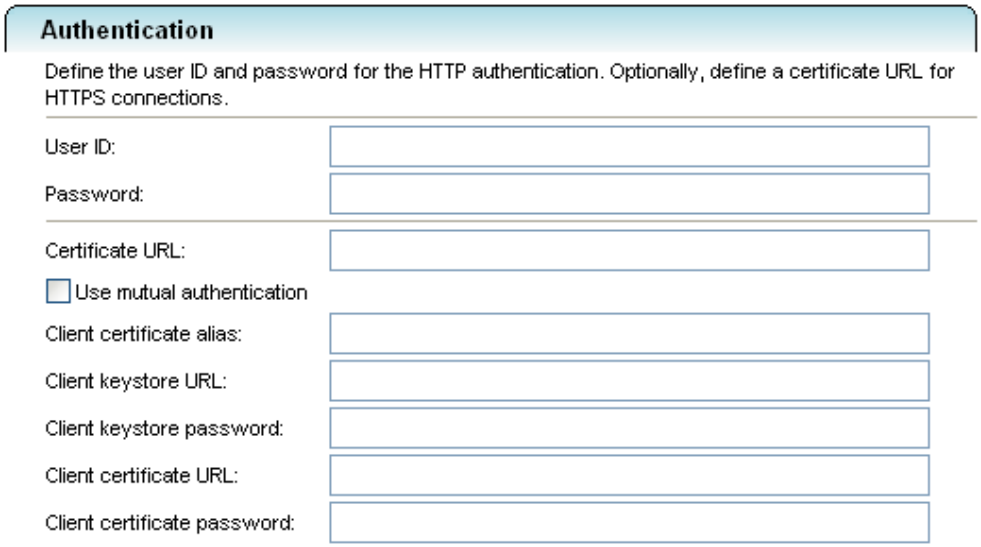


More information about XPath can be found at http://www.w3.org/TR/xpath.

Copyright © 2001-2011 Voxeo Germany GmbH

Authentication

In the **Authentication** section the user ID and password for the HTTP authentication are specified, plus an optional certificate URL for HTTPS connections and several settings regarding mutual authentication.



If authentication fails, a corresponding HTTP return code is made available through the return value of the Connector item definition. At the same time, an *Error – Connector* event is activated.

Property	Description
User ID	The user ID used for authentication. Note that **User ID** can also be defined using e.g. Variable objects that are filled by an external configuration using application defaults. See the corresponding section in Chapter 4 – *Service Deployment* of the *Deployment Guide* for details.
Password	The password used for authentication. Note that **Password** can also be defined using e.g. Variable objects that are filled by an external configuration using application defaults. See the corresponding section in Chapter 4 – *Service Deployment* of the *Deployment Guide* for details.
Certificate URL	URL pointing to the certificate to be used for the HTTPS communication (required when using HTTPS). This URL may itself only use the HTTP or FILE protocol (but not HTTPS). Note that if more than one Connector item is specified in the Connector item list of the Connector object, the same certificate URL is applied to all items. If different certificates are required, different Connector objects must be defined.
Use mutual authentication	Indicates whether mutual authentication should be used. If enabled, a client keystore and certificate along with their respective passwords need to be defined either in the object definition or by use of system properties.

Copyright © 2001-2011 Voxeo Germany GmbH

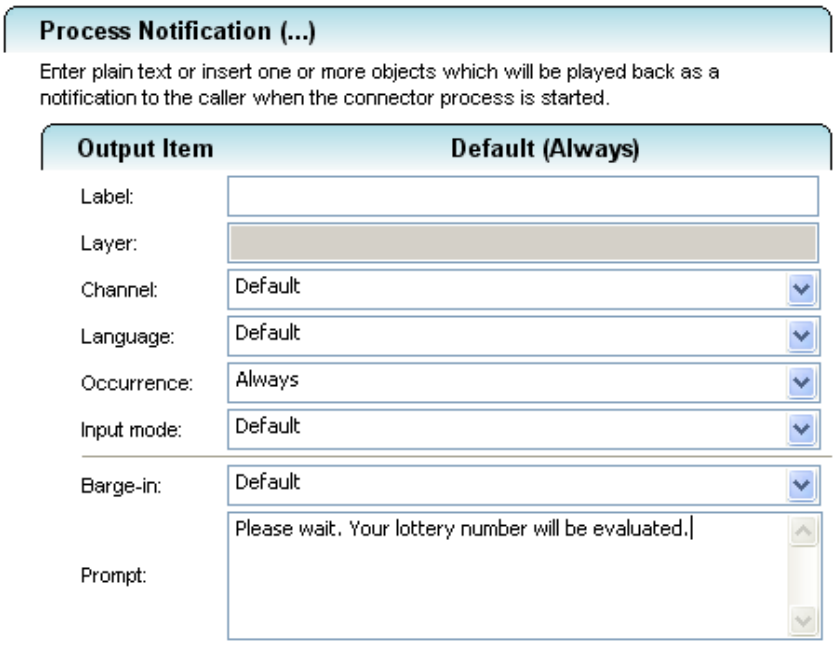
Property	Description
Client certificate alias	Defines the alias by which the client certificate can be retrieved from the client keystore. See below this table for a description of the recommended mechanism.
Client keystore	Defines the URL to the client keystore. While in principle you may use multiple keystore, it is recommended to use a single one. The client keystore can also be set using the system property *https.client.keystore.url*, which allows use of the overwrite mechanism described in the section Dynamic Property Retrieval in Chapter 1 – *Advanced Configuration* in the *Administration Guide*.
Client keystore password	Defines the password for the client keystore. The client keystore password can also be set using the system property *https.client.keystore.pwd*, which allows use of the overwrite mechanism described in the section Dynamic Property Retrieval in Chapter 1 – *Advanced Configuration* in the *Administration Guide*.
Client certificate URL	Defines the URL to the client certificate. Alternatively, the **Client certificate alias** can be used to reference the certificate within a keystore. The client certificate can also be set using the system property *https.client.certificate.url*, which allows use of the overwrite mechanism described in the section Dynamic Property Retrieval in Chapter 1 – *Advanced Configuration* in the *Administration Guide*.
Client certificate password	Defines the password for the client certificate. The client keystore can also be set using the system property *https.client.keystore.pwd*, which allows use of the overwrite mechanism described in the section Dynamic Property Retrieval in Chapter 1 – *Advanced Configuration* in the *Administration Guide*.

The recommended mechanism for mutual authentication is to work with a single client keystore in which individual certificates are referenced using the **Client certificate alias**. This simplifies management and maintenance of both the installation and the application.

Process Notification

In the **Process Notification** section, an embedded Output object can be specified that gives the caller a notification that the connector process is starting. This parameter is optional.

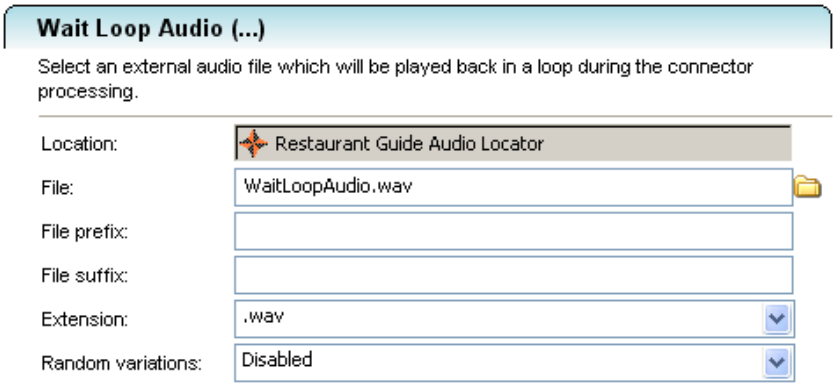
Copyright © 2001-2011 Voxeo Germany GmbH



For further details, refer to the Output object in this *Object Reference*.

Wait Loop Audio

In the **Wait Loop Audio** section, an audio file can be specified that is played back while the connector process is running. This parameter is optional and only applies to the voice and video channel. It is typically used for time-consuming connector operations, e.g. to play hold music.



For further details on the possible settings, refer to the Audio object in this *Object Reference*.

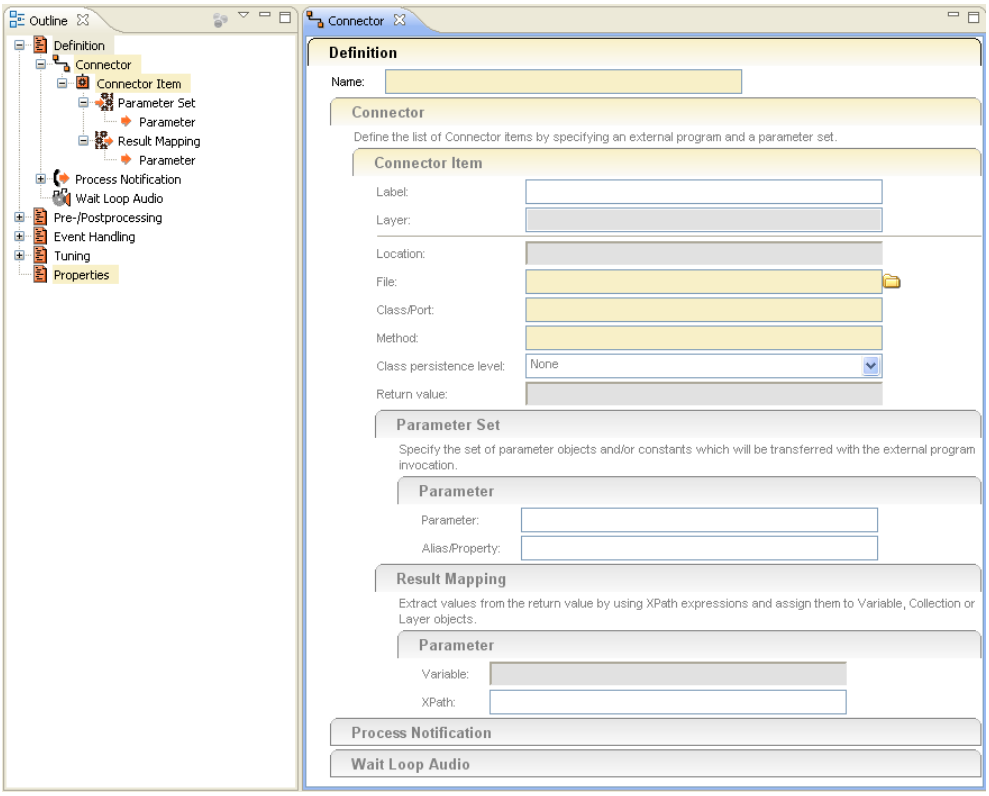
Note: On some media platforms, the duration of the audio file must be at least as long as the processing time needed by the Connector object. On other platforms, an audio file that is shorter than the processing time required by the connector will be looped continuously as needed. For further information on the behavior of audio processing refer to the local documentation of the media platform vendor.

Copyright © 2001-2011 Voxeo Germany GmbH

Java Connector Definition

The **Definition** of a Java Connector object provides the following sections:

- Connector
  To define one or more Connector items.
- Process Notification
  To optionally specify a notification to the caller that the connector processing has started. For details refer to the Process Notification description of the HTTP Connector.
- Wait Loop Audio
  To optionally specify an audio file that is played back while the connector process is running. For details refer to the Wait Loop Audio description of the HTTP Connector.

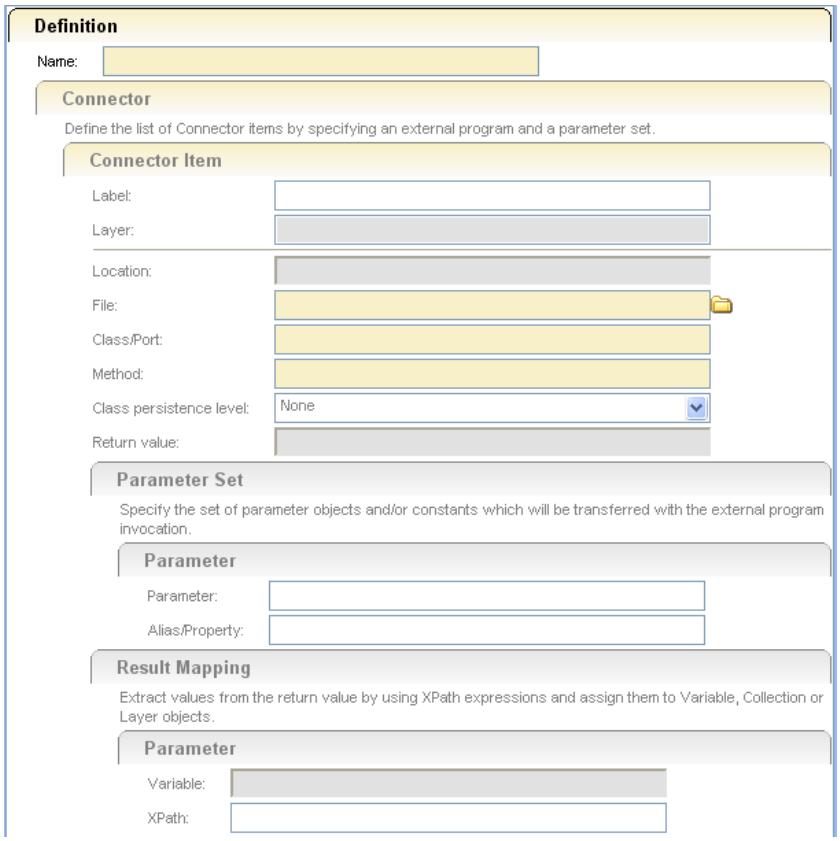


For further details regarding additional object configuration, refer to *Pre-/Postprocessing*, *Event Handling*, *Tuning*, or *Properties* in this *Object Reference*.

Connector

In the **Connector** section, define at least one **Connector Item** that will process a Java program.

Copyright © 2001-2011 Voxeo Germany GmbH



Each **Connector Item** contains a set of fields:

Property	Description
Label	Optional parameter to identify the Connector item in a list.
Backend ID	Optional parameter to identify the back-end component that is accessed by the Connector item.
Layer	Optional parameter to indicate the layer that the Connector item belongs to. For more information on layers, see Chapter 7 – *How to Use Layers* in the *Design Guide* or the Layer object in this *Object Reference*.
Location	Defines the URL where the corresponding JAR files are located. Note that JAR files cannot be retrieved using HTTPS URLs; only HTTP and file URLs are supported.
File	Specifies the name of the JAR file. It is also possible to define a comma-separated list of JAR files.
Class	Defines the name of the class, prefixed by its package name, which contains the method to call.
Method	Defines the method to be called.
Class persistence level	Indicates if class instances of Connector items are to be persisted, either at dialog, service or server instance level: • *None* The server creates a new class instance every time this

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
	Connector item is processed. • *Dialog* The server creates a class instance the first time the Connector item is processed and then persists this instance for any subsequent processing of this Connector item during the current dialog session. • *Service* The server creates a class instance the first time any dialog for this service processes this Connector item and then persists this instance at service level. • *Server Instance* The server creates a class instance the first time any dialog for any service uses the JAR file and class defined in the Connector item; the instance is then persisted for any subsequent processing in any dialog of any service. By default, the **Class persistence level** option is set to *None* and a new class instance is created every time the Connector item is processed.  **Caution:** If a connector instance is shared between multiple calls on service or server level an instance of the connector class is stored on that level and shared between multiple threads (one per call). This means that there is concurrency on instance and not just on class level. The implementer of the connector code has the responsibility to ensure thread-safety, which means that the connector should be either stateless or use appropriate synchronization mechanisms to ensure consistency of the state at any time.
Return value	Allows to link a Variable, Collection, or Layer object which will be set to the return value of the method called. If the return type of the method is *void*, the return value is empty and the linked object will not be overwritten.

The individual Connector items are processed one after the other in the order in which they are defined. The failure of any one of these can be handled by defining event handling for the event type *Error – Connector*. For more information, refer to *Event Handling* in this *Object Reference*. If no such event handling is defined and a Connector item fails, an error message is played and the dialog ends.

Note: When using Java code within a Connector item, it needs to be kept in mind that this Java code is executed within the same JVM as VoiceObjects Server. Thus the performance of the custom Java code may negatively impact the performance of the server. This also concerns such issues as garbage collection if the custom code creates large numbers of objects.

By default, when calling Java code within a Connector item, it is required to restart the JVM which runs the server to enforce changes made to the JAR file. This can be modified in the configuration file by setting `<monitorJarFiles>` to *true* (the default is *false*). The location for the configuration file is:

../VoiceObjects/Platform/WEB_INF/config/components_VOServer.xml

Copyright © 2001-2011 Voxeo Germany GmbH

If set to *true*, JAR files are dynamically reloaded if they are referenced via relative file paths denoted as

file:../Resources/

where the relative path starts at the *../VoiceObjects/Platform/WEB-INF directory*.

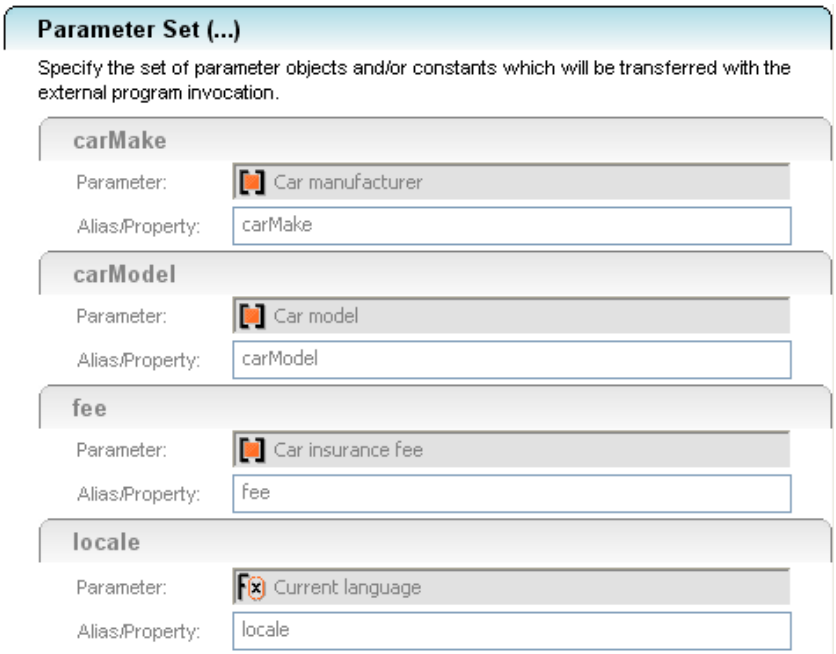
Dynamic JAR reloading using absolute paths only works on Microsoft Windows. Absolute file paths need to be denoted as

file:/C:/MyJavaCode/

Caution: Using dynamic JAR reloading is discouraged in production environments. Moreover, it only works under certain circumstances. Under Windows, a file cannot be copied or moved over, as an open file handle prevents that. Under UNIX, a file can be copied or moved over, but it will not be reloaded in this case. So in all cases, make sure to copy or move the JAR to a new filename, adjust the resource locator definition in the Connector object and redeploy the application. In addition, dynamic JAR reloading does not work when the JAR file is referenced using an HTTP or file URI.

Parameter Set

The **Parameter Set** section of a **Connector Item** enables the exchange of processing information with the Java code. Possible parameter types are constants, Resource Locator, Variable, Collection, Expression, Layer, and Script objects. The parameter set is optional.



Any number of parameters can be exchanged with a connector.

The name of a class property (e.g. carMake), which has to be defined based on the Java Bean specification, needs to be specified in the **Alias/Property** field. In order for the value to be passed to the Java code, a corresponding set method (e.g. setCarMake()) needs to be provided, which is called before the Connector item method is processed. If the Java code also provides a get method (e.g. getCarMake()) for this property, and the entry in the **Parameter** field is a Variable, Collection or Layer object, this get method is called after the Connector item method has been executed,

Copyright © 2001-2011 Voxeo Germany GmbH

and the resulting value is assigned to the respective Variable, Layer, or Collection object. Special get and set methods can be provided to pass error codes and messages to the server. See Error Handling for more details on this.

Both the get and the set methods need to be of type *public String*.

Note: Collection objects are passed to the Java code as an XML string that needs to be interpreted within the Java code itself. Collection values being passed back must also be provided (using the respective getter method) in this XML format. For a definition of the format itself, see the Collection object in this *Object Reference*.

When using special XML characters within a parameter (like ampersand “&”), make sure to mask them using the correct XML entity (e.g. “&” for “&”).

Result Mapping

The **Result Mapping** section allows extracting specific values by applying an XPath expression to the return value. Any number of result mappings can be defined. The expression specified in the **XPath** field will be applied to the return value of the connector. The result of this operation will be assigned to the Variable, Collection or Layer object defined in the **Variable** field. As XPath is designed to retrieve values from a well-formed XML structure, the Connector item must return such a value. If the Connector item returns nothing or a non-XML value the *Error – Internal* event is activated.

The **Return Value** field must not necessarily contain a Variable, Collection or Layer object. The result mapping items will be assigned nevertheless. If this field contains an object, the unchanged return value will be assigned to it.

Example:

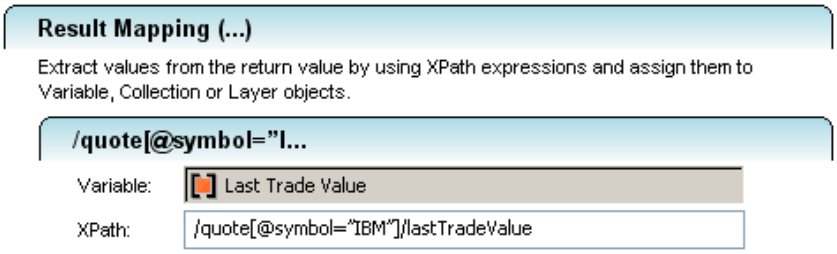
Given a Connector item that returns the following XML structure:

```
<quote symbol="IBM">
    <lastTradeValue>82.17</lastTradeValue>
    <previousCloseValue>82.16</previousCloseValue>
</quote>
```

The following XPath expression can be used to retrieve the last trade value:

```
/quote[@symbol="IBM"]/lastTradeValue
```

The **Result Mapping** section in the **Connector** editor would look like this:



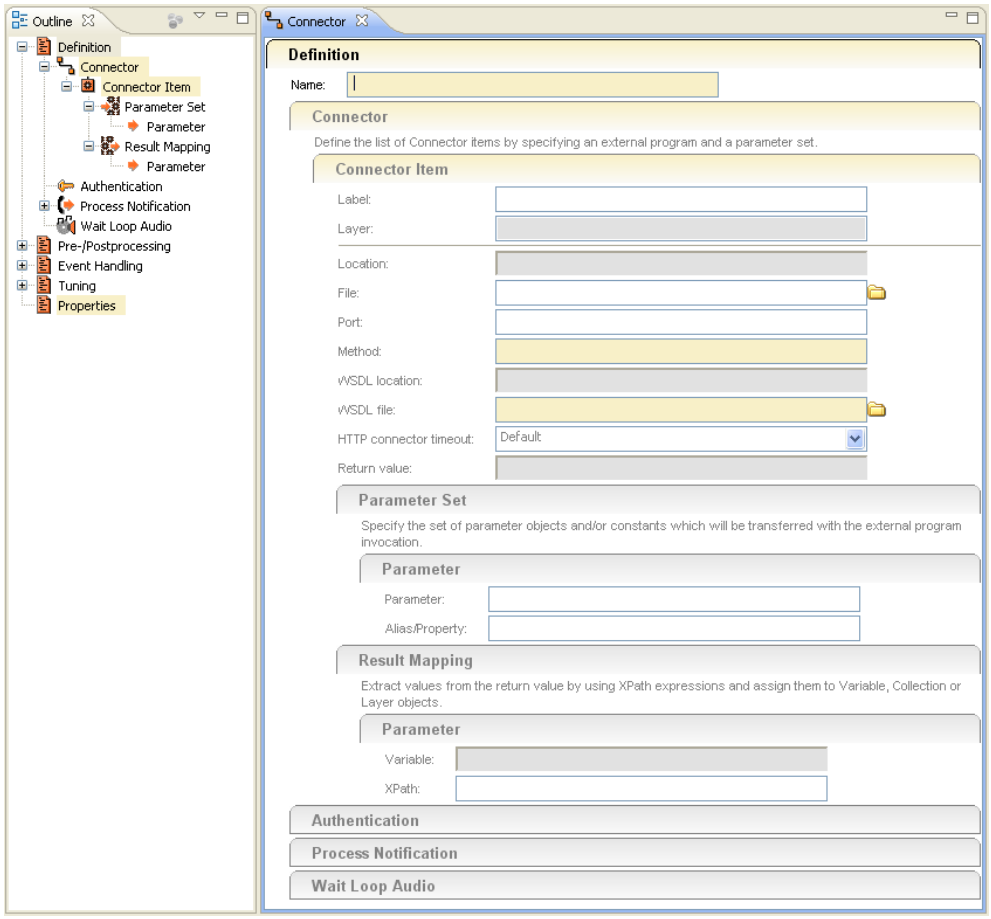
More information about XPath can be found at http://www.w3.org/TR/xpath.

Copyright © 2001-2011 Voxeo Germany GmbH

Web Service Connector

The **Definition** of a Web Service Connector object provides the following sections:

- Connector
  To define one or more Connector items.
- Authentication
  To define the user ID and password for HTTP authentication, and a certificate URL for HTTPS connections. For details refer to the Authentication description of the HTTP Connector.
- Process Notification
  To optionally specify a notification to the caller that the connector processing has started. For details refer to the Process Notification description of the HTTP Connector.
- Wait Loop Audio
  To optionally specify an audio file that is played back while the connector process is running. For details refer to the Wait Loop Audio description of the HTTP Connector.



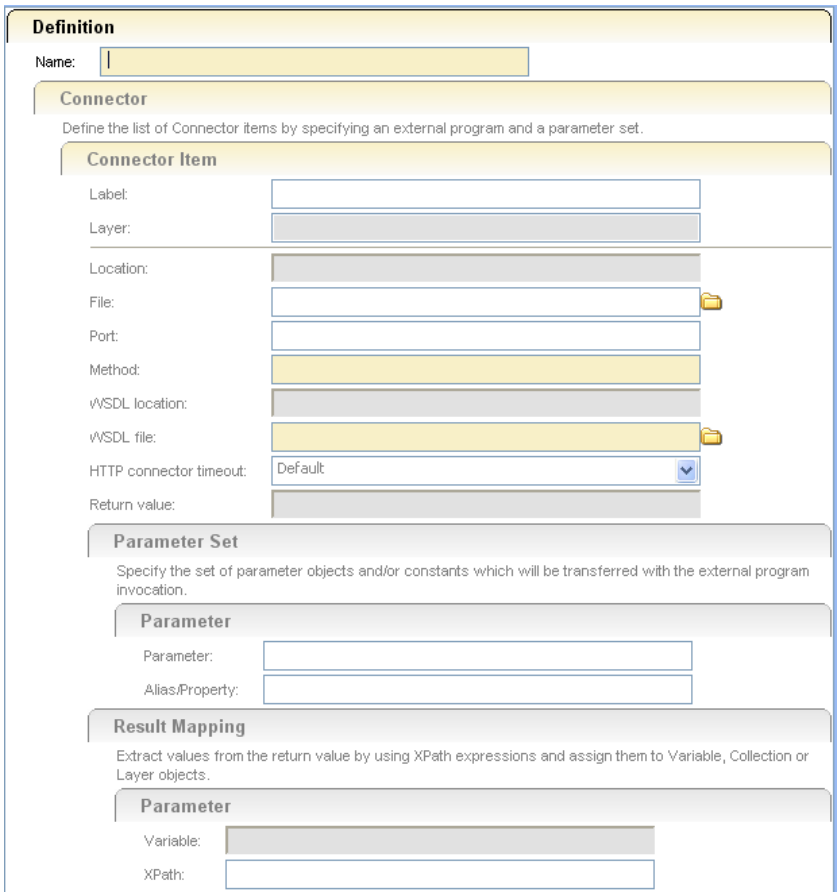
For further details regarding additional object configuration, refer to *Pre-/Postprocessing*, *Event Handling*, *Tuning*, or *Properties* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Connector

In the **Connector** section, define at least one **Connector Item** that will process a Web services call.

Note, that with Web Service connectors, only document-style SOAP Web Services are supported. Web Service connectors do not support RPC-style Web Services. If you are calling an RPC-style Web Service, you need to use a custom Java or CGI connector instead, which will call the Web Service method itself and process the return data appropriately.



Each **Connector Item** contains a set of fields:

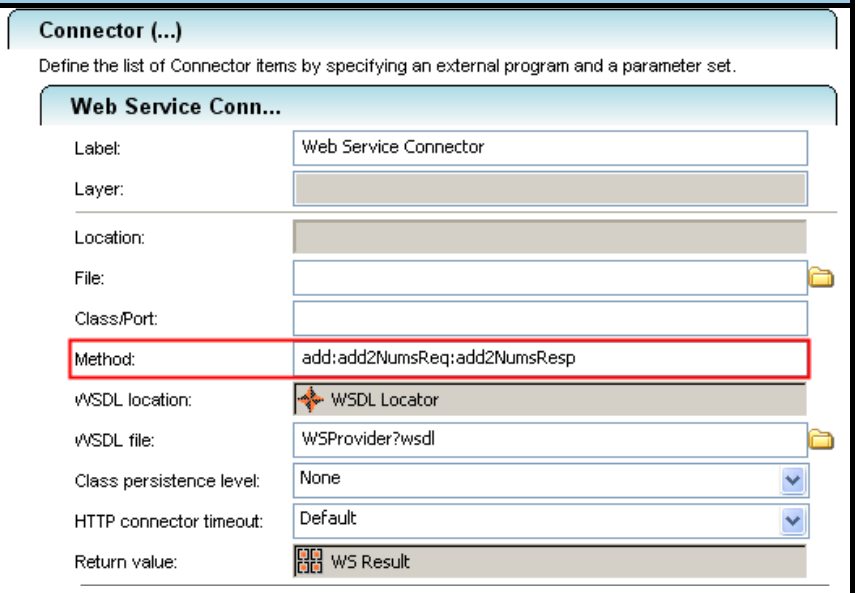
Property	Description
Label	Optional parameter to identify the Connector item in a list.
Backend ID	Optional parameter to identify the back-end component that is accessed by the Connector item.
Layer	Optional parameter to indicate the layer that the Connector item belongs to. For more information on layers, see Chapter 7 – *How to Use Layers* in the *Design Guide* or the Layer object in this *Object Reference*.
Location	Defines the URL of the service provider node, in case it is different from the location of the WSDL file. Typically, the location of the service provider node is the same as that of the WSDL file, in which

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
	case you can leave this field empty.
File	Specifies the name of the service provider node, in case it is different from the name of the WSDL file. Typically, the location of the service provider node is the same as that of the WSDL file, in which case you can leave this field empty.
Port	Defines the name of the port to be used. If the WSDL file provides only one port, this field can be left empty.

Method	Defines the Web service method to be called. If there is more than one method with the same name available at the service endpoint, these methods are distinguished by the name of the input and output message. If this is the case, append the names of the input and output message to the **Method** field, using colons as separators.
	Example: The method to be called is "add". There are two versions available; the WSDL description of the first version is
	`<wsdl:operation name="add" parameterOrder="num1 num2">`
	`<wsdl:input message="impl:add2NumsRequest" name="add2NumsReq"/>`
	`<wsdl:output message="impl:add2NumsResponse" name="add2NumsResp"/>`
	`</wsdl:operation>`
	whereas the description of the second version is
	`<wsdl:operation name="add" parameterOrder="num1 num2 num3">`
	`<wsdl:input message="impl:add3NumsRequest" name="add3NumsReq"/>`
	`<wsdl:output message="impl:add3NumsResponse" name="add3NumsResp"/>`
	`</wsdl:operation>`
	If the first version is to be called, the **Method** field looks as follows:

Copyright © 2001-2011 Voxeo Germany GmbH

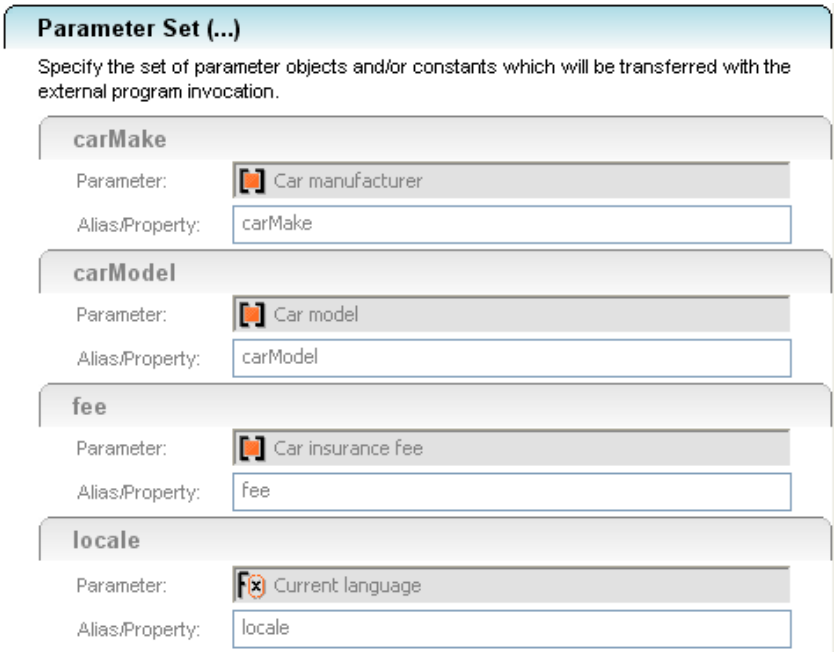
Property	Description
	
WSDL location	Defines the URL of the WSDL file that describes the available Web services of the service provider.
WSDL file	Defines the name of the WSDL file.
HTTP connector timeout	Specifies the time after which the request is aborted if no answer is received from the back-end. If the connector takes longer to process than the time specified in this field, an *Error – Connector* event is activated. You can select one of the values of the drop-down list or enter a custom value. Custom values are interpreted as seconds. If you leave it at *Default*, the setting from the corresponding Service object definition is taken. See Processing the Connector Object for interactions with the VoiceXML *fetchtimeout* property.
Return value	Allows to link a Variable, Collection, or Layer object which will be set to the return value of the connector, which is the content of the response. Variable objects are typically used to store the result. Apply the *XPath* function of the Expression object to examine the result, if desired, or directly use **Result Mapping**, as described below. Only the first XML node (including all of its child nodes) of the response is accessible for VoiceObjects. The content of this first node is accessible as the return value.

The individual Connector items are processed one after the other in the order in which they are defined. The failure of any one of these can be handled by defining event handling for the event type *Error – Connector*. For more information, refer to *Event Handling* in this *Object Reference*. If no such event handling is defined and a Connector item fails, an error message is played and the dialog ends.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter Set

The **Parameter Set** section of a **Connector Item** enables the exchange of processing information with the Web service. Possible parameter types are constants, Resource Locator, Variable, Collection, Expression, Layer, and Script objects. The parameter set is optional.



Note that the parameter set only defines input parameters for the Web service; return values can only be accessed through the **Return value** object.

The exact usage of the parameter set depends on the type of input parameters that are expected by the Web service (based on the WSDL). If the input parameters consist of a sequence of simple types, each of the input parameters can be specified as a separate item in the parameter set. If one of the input parameters has a complex type, only one item is defined in the parameter set. In this case the single parameter contains the complete XML structure (the request message) that is part of the SOAP body. This XML structure will directly be copied into the SOAP body. The following two examples show the details for both scenarios.

Example 1 (with simple types):

This is an example for a WSDL with an operation that only uses simple data types as input parameters.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://www.voiceobjects.com/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://www.voiceobjects.com/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://www.voiceobjects.com/">
      <s:element name="MyOperationRequest">
        <s:complexType>
```

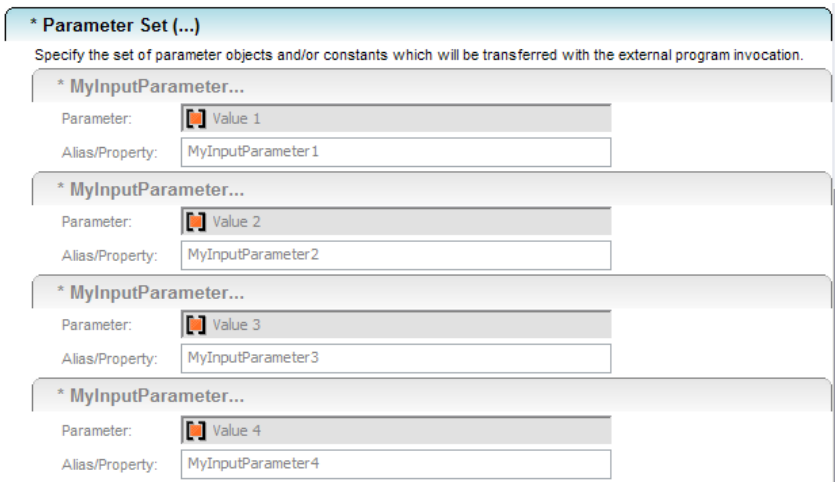
Copyright © 2001-2011 Voxeo Germany GmbH

```
            <s:sequence>
              <s:element minOccurs="0" maxOccurs="1"
                name="MyInputParameter1" type="s:string" />
              <s:element minOccurs="0" maxOccurs="1"
                name="MyInputParameter2" type="s:string" />
              <s:element minOccurs="0" maxOccurs="1"
                name="MyInputParameter3" type="s:string" />
              <s:element minOccurs="0" maxOccurs="1"
                name="MyInputParameter4" type="s:string" />
            </s:sequence>
          </s:complexType>
        </s:element>
        <s:element name="MyOperationResponse">
          <s:complexType>
            <s:sequence>
              <s:element minOccurs="0" maxOccurs="1"
                name="MyOperationResult" type="tns:ResponseData" />
            </s:sequence>
          </s:complexType>
        </s:element>
        <s:complexType name="ResponseData">
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
                name="MyResultValue1" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1"
                name="MyResultValue2" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1"
                name="MyResultValue3" type="s:string" />
            <s:element minOccurs="1" maxOccurs="1"
                name="MyResultValue4" type="s:double" />
            <s:element minOccurs="0" maxOccurs="1"
                name="MyResultValue5" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:schema>
    </wsdl:types>
    <wsdl:message name="MyOperationSoapIn">
      <wsdl:part name="parameters" element="tns:MyOperationRequest" />
    </wsdl:message>
    <wsdl:message name="MyOperationSoapOut">
      <wsdl:part name="parameters" element="tns:MyOperationResponse" />
    </wsdl:message>
    <wsdl:portType name="MyServiceSoap">
      <wsdl:operation name="MyOperation">
        <wsdl:input message="tns:MyOperationSoapIn" />
        <wsdl:output message="tns:MyOperationSoapOut" />
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="MyServiceSoap" type="tns:MyServiceSoap">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
      <wsdl:operation name="MyOperation">
        <soap:operation
soapAction="http://www.voiceobjects.com/MyOperation" style="document"
/>
        <wsdl:input>
          <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
          <soap:body use="literal" />
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="MyService">
      <wsdl:port name="MyServiceSoap" binding="tns:MyServiceSoap">
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
      <soap:address
location="http://www.voiceobjects.com/ws/MyService.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

The operation `MyOperation` is expecting an input structure `MyOperationRequest` that is a sequence of strings (i.e. simple types). The corresponding parameter set would look like this:



Example 2 (with complex types):

This is an example for a WSDL with an operation that uses a complex type as input parameter.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://www.voiceobjects.com/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://www.voiceobjects.com/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://www.voiceobjects.com/">
      <s:element name="MyOperationRequest">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
              name="MyInputParameter1" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1"
              name="MyInputParameter2" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1"
              name="MyInputParameter3" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1"
              name="MyInputParameter4" type="tns:ComplexParameter" />
          </s:sequence>
        </s:complexType>
      <s:element name="ComplexParameter">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="A"
```

Copyright © 2001-2011 Voxeo Germany GmbH

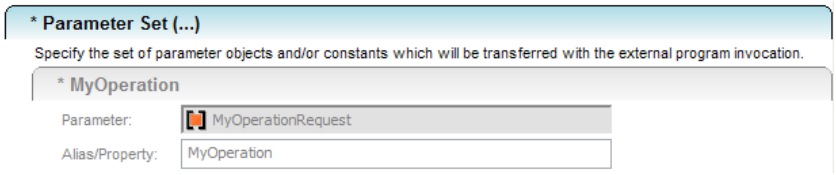
```
                type="s:string" />
              <s:element minOccurs="0" maxOccurs="1" name="C"
                type="s:string" />
              <s:element minOccurs="0" maxOccurs="1" name="C"
                type="s:string" />
            </s:sequence>
          </s:complexType>
        </s:element>
        <s:element name="MyOperationResponse">
          <s:complexType>
            <s:sequence>
              <s:element minOccurs="0" maxOccurs="1"
                name="MyOperationResult" type="tns:ResponseData" />
            </s:sequence>
          </s:complexType>
        </s:element>
        <s:complexType name="ResponseData">
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
                name="MyResultValue1" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1"
                name="MyResultValue2" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1"
                name="MyResultValue3" type="s:string" />
            <s:element minOccurs="1" maxOccurs="1"
                name="MyResultValue4" type="s:double" />
            <s:element minOccurs="0" maxOccurs="1"
                name="MyResultValue5" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:schema>
    </wsdl:types>
    <wsdl:message name="MyOperationSoapIn">
      <wsdl:part name="parameters" element="tns:MyOperationRequest" />
    </wsdl:message>
    <wsdl:message name="MyOperationSoapOut">
      <wsdl:part name="parameters" element="tns:MyOperationResponse" />
    </wsdl:message>
    <wsdl:portType name="MyServiceSoap">
      <wsdl:operation name="MyOperation">
        <wsdl:input message="tns:MyOperationSoapIn" />
        <wsdl:output message="tns:MyOperationSoapOut" />
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="MyServiceSoap" type="tns:MyServiceSoap">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
      <wsdl:operation name="MyOperation">
        <soap:operation
soapAction="http://www.voiceobjects.com/MyOperation" style="document"
/>
        <wsdl:input>
          <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
          <soap:body use="literal" />
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="MyService">
      <wsdl:port name="MyServiceSoap" binding="tns:MyServiceSoap">
        <soap:address
location="http://www.voiceobjects.com/ws/MyService.asmx" />
      </wsdl:port>
    </wsdl:service>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
</wsdl:definitions>
```

In this case the operation `MyOperation` is expecting an input structure `MyOperationRequest` that contains a complex data type (`MyInputParameter4`). The complete `MyOperationRequest` XML structure is defined as one single parameter in the parameter set.

The **Parameter Set** section in the **Connector** editor would look like this:



There is only one single parameter in the parameter set and the Variable object *MyOperationRequest* contains the following XML structure:

```
<ns1:MyOperationRequest xmlns:ns1="http://www.voiceobjects.com">
    <ns1:MyInputParameter1>value 1</ns1:MyInputParameter1>
    <ns1:MyInputParameter1>value 2</ns1:MyInputParameter1>
    <ns1:MyInputParameter1>value 3</ns1:MyInputParameter1>
    <ns1:MyInputParameter4>
        <ns1:ComplexParameter>
            <ns1:A>value A</ns1:A>
            <ns1:B>value B</ns1:B>
            <ns1:C>value C</ns1:C>
        </ns1:ComplexParameter>
    </ns1:MyInputParameter1>
</ns1:MyOperationRequest>
```

Result Mapping

The **Result Mapping** section allows extracting specific values by applying an XPath expression to the return value. Any number of result mappings can be defined. The expression specified in the **XPath** field will be applied to the return value of the connector. The result of this operation will be assigned to the Variable, Collection or Layer object defined in the **Variable** field. As XPath is designed to retrieve values from a well-formed XML structure, the Connector item must return such a value. If the Connector item returns nothing or a non-XML value the *Error – Internal* event is activated.

The **Return Value** field must not necessarily contain a Variable, Collection or Layer object. The result mapping items will be assigned nevertheless. If this field contains an object, the unchanged return value will be assigned to it.

Example:

Given a Connector item that returns the following XML structure:

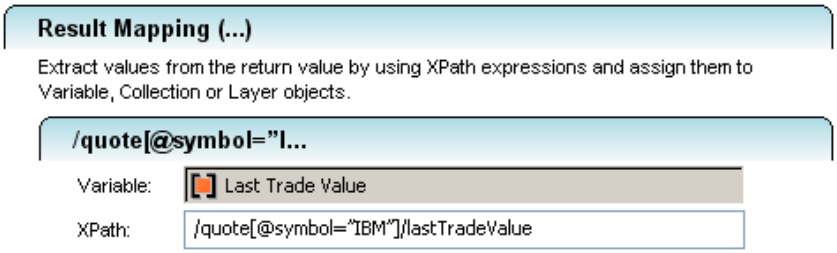
```
<quote symbol=”IBM”>
    <lastTradeValue>82.17</lastTradeValue>
    <previousCloseValue>82.16</previousCloseValue>
</quote>
```

The following XPath expression can be used to retrieve the last trade value:

```
/quote[@symbol=”IBM”]/lastTradeValue
```

Copyright © 2001-2011 Voxeo Germany GmbH

The **Result Mapping** section in the **Connector** editor would look like this:



When calling a Web service, the XML structure that is returned by the service (as defined in the WSDL) is assigned to the return value and can be accessed using the **Result Mapping**.

In the WSDL examples above a return value can look like this example structure.

```
<MyOperationResult xmlns="http://www.voiceobjects.com/">
    <MyResultValue1>Some value</MyResultValue1>
    <MyResultValue2>Some other value</MyResultValue2>
</MyOperationResult>
```

In this example the following XPath expression can be used to retrieve the value of `MyResultValue1`:

```
/MyOperationResult/MyResultValue1
```

Only the first XML node (including all of its child nodes) of the response is accessible for VoiceObjects. The content of this first node is the top-level node for the XPath expressions (`MyOperationResult` in the example above).

More information about XPath can be found at http://www.w3.org/TR/xpath.

Processing the Connector Object

This section contains some technical remarks on the treatment of Connector objects in VoiceObjects Server.

For each Connector object used in the voice or video channel, the server sends a VoiceXML document (a single one for all Connector items) to the media platform, even though there is no interaction with the caller. The document consists of a `<submit>` element including the *fetchaudio* attribute. If a wait loop audio is specified for the object, the audio is mentioned in this attribute, so that the platform plays this file while the connector is processed. Any output defined before the Connector object (either in the **Process Notification** section within the object, or in Output objects prior to the Connector object in a sequence) will be "flushed" by the platform, i.e. played back before the wait loop audio is applied, which is a desirable effect. To achieve this effect even if no wait loop audio is defined, the server renders a reference to an empty audio file (taken from the system resource folder of the VoiceObjects installation) in the *fetchaudio* attribute. Without this "flush" effect, a media platform might withhold the output defined prior to the Connector object until the connector has been processed, which would result in undesirable delays.

At the same time, a *fetchtimeout* attribute is rendered and set to the value of *10000s* (ten thousand seconds). The effect of this is that the media platform is forced to wait until the connector has been processed and does not try to request another VoiceXML document from the server before the connector has been processed. If it did, the server would reject the request. Note that due to this, any user-defined setting of *fetchtimeout* (e.g. through the tuning property *Resource Fetching – Fetch Timeout*) will

Copyright © 2001-2011 Voxeo Germany GmbH

be ignored for CGI and Web Service connectors. If you want to define a timeout value for these connectors, use the **HTTP connector timeout** setting instead.

In the text and Web channel, there will be no intermediate page rendered, unless there is a process notification defined which is rendered as a standalone output.

Exchanging Data via an XML API

In the case of HTTP Connectors, data is usually exchanged by means of an XML API. There are two different types of XML:

- Request XML
- Result XML

For simpler cases, there is the lightweight **Simple HTTP Mode** in which parameters are passed as normal HTTP POST parameters, and a single return value can be provided by the CGI script. This return value may either be a string, or a collection structure that can then also use **Result Mapping** to be inspected and assigned to other variables.

Request XML

The Request XML is generated by the server and contains processing information like the dialog ID (a globally unique string created by the server that identifies the dialog session), the ANI (Automated Number Identification), the DNIS (Dialed Number Identification Service) and the optional parameters from the parameter set.

The Request XML is sent via the HTTP connection method *POST*. The CGI program needs to extract the request information string via the request parameter *OVAPRequest*. VoiceObjects provides a framework for Java Servlets that encapsulates the extraction of the request parameter (refer to CGI Connector Framework for further details).

The XML consists of a single `<root> … </root>` block containing any number of rows encapsulated in `<row> … </row>` tags. Each row contains one parameter to be transmitted to the CGI script. Within each row, there are four columns providing the relevant aspects of the parameter:

- `<col name=″type″> … </col>`
  Provides the type of the parameter, which can be *expression*, *variable*, *collection*, *layer*, or *constant*. Note that Resource Locator and Script objects are categorized under *expression* (see examples below).
- `<col name=″name″> … </col>`
  Provides the name of the object that is passed. This column is empty for constants.
- `<col name=″referenceID″> … </col>`
  Provides the reference ID or the alias (if defined) of the object that is passed. For constants, it provides the alias specified in the **Alias/Property** field of the parameter set.
- `<col name=″value″> … </col>`
  Provides the value of the object that is passed, wrapped in CDATA[]. In the case of collections, this value is an XML structure itself. See the Collection object in this *Object Reference* for a definition of the structure.

The XML always contains general information about the dialog, including the dialog ID, the ANI, and the DNIS.

Copyright © 2001-2011 Voxeo Germany GmbH

Below is an example of a Request XML with an empty parameter set definition:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<request>
    <root>
        <row>
            <col name="type">expression</col>
            <col name="name">dialogID</col>
            <col name="referenceID">dialogID</col>
            <col name="value"><![CDATA[OVAPac16172d00000
            0000000058e000000f490ffef8b]]></col>
        </row>
        <row>
            <col name="type">expression</col>
            <col name="name">ANI</col>
            <col name="referenceID">ANI</col>
            <col name="value"><![CDATA[12345]]></col>
        </row>
        <row>
            <col name="type">expression</col>
            <col name="name">DNIS</col>
            <col name="referenceID">DNIS</col>
            <col name="value"><![CDATA[4085371847]]></col>
        </row>
    </root>
</request>
```

Any number of constants or Variable, Expression, Collection, Script or Layer objects can be transmitted to the script. To transfer them to the CGI script, they need to be defined in the parameter set of the Connector object.

Each constant in the parameter set creates a block of the following form in the Request XML:

```
<row>
       <col name="type">constant</col>
       <col name="name"></col>
       <col name="referenceID">YearOfBirth</col>
       <col name="value"><![CDATA[1957]]></col>
</row>
```

Note that constants do not have names. The alias defined in the parameter set is provided in the column *referenceID*.

Copyright © 2001-2011 Voxeo Germany GmbH

Each Variable object in the parameter set creates a block of the following form in the Request XML:

```
<row>
	<col name="type">variable</col>
	<col name="name">MyVariable</col>
	<col name="referenceID">myvariable</col>
	<col name="value"><![CDATA[foobar]]></col>
</row>
```

Each Expression object in the parameter set creates a block of the following form in the Request XML:

```
<row>
	<col name="type">expression</col>
	<col name="name">MyExpression</col>
	<col name="referenceID">myexpression</col>
	<col name="value"><![CDATA[17]]></col>
</row>
```

Each Collection object in the parameter set creates a block of the following form in the Request XML:

```
<row>
	<col name="type">collection</col>
	<col name="name">MyCollection</col>
	<col name="referenceID">mycollection</col>
	<col name="value"><![CDATA[<root/>]]></col>
</row>
```

Each Script object in the parameter set creates a block of the following form in the Request XML:

```
<row>
	<col name="type">expression</col>
	<col name="name">MyScript</col>
	<col name="referenceID">myscript</col>
	<col name="value"><![CDATA[The result of the script.]]></col>
</row>
```

Each Layer object in the parameter set creates a block of the following form in the Request XML:

```
<row>
	<col name="type">layer</col>
	<col name="name">MyLayer</col>
	<col name="referenceID">mylayer</col>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
        <col name="value">
                <![CDATA[The active states of the layer.]]>
        </col>
</row>
```

Each Resource Locator object in the parameter set creates a block of the following form in the Request XML (note that only the URL is passed to the connector):

```
<row>
        <col name="type">expression</col>
        <col name="name">MyResourceLocator</col>
        <col name="referenceID">mylocator</col>
        <col name="value"><![CDATA[http://somewhere.com/]]></col>
</row>
```

The following is an example of a request with a Variable object and an Expression object added to the parameter set.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<request>
    <root>
        <row>
            <col name="type">expression</col>
            <col name="name">dialogID</col>
            <col name="referenceID">dialogID</col>
            <col name="value"><![CDATA[OVAPac16172d000000000
            000058e000000f490ffef8b]]></col>
        </row>
        <row>
            <col name="type">expression</col>
            <col name="name">ANI</col>
            <col name="referenceID">ANI</col>
            <col name="value"><![CDATA[12345]]></col>
        </row>
        <row>
            <col name="type">expression</col>
            <col name="name">DNIS</col>
            <col name="referenceID">DNIS</col>
            <col name="value"><![CDATA[4085371847]]></col>
        </row>
        <row>
            <col name="type">variable</col>
            <col name="name">Variable</col>
            <col name="referenceID">myvariable</col>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
            <col name="value">abcde</col>
        </row>
        <row>
            <col name="type">expression</col>
            <col name="name">Expression</col>
            <col name="referenceID">myexpression</col>
            <col name="value"><![CDATA[12]]></col>
        </row>
    </root>
</request>
```

Result XML

The CGI script may assign return values to any (or all) of the Variable, Layer, or Collection objects transmitted to it. Note that it is not possible to assign a return value to a Variable, Layer, or Collection object that is not part of the parameter set. It is also not possible to assign return values to constants or Expression and Script objects. To assign a return value to a Variable, Layer, or Collection object, the respective reference ID or alias is used.

For each assignment, two lines in the result XML are required: One to provide the reference ID of the respective Variable, Layer, or Collection object, and one to provide the value.

As an example, the following XML assigns *128* to the variable with reference ID *myvariable*, and the table ((*John,Smith*),(*Bob,Miller*)) to the collection with reference ID *mycollection*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
    <root>
        <row>
            <col name="referenceID">myvariable</col>
            <col name="value">128</col>
        </row>
        <row>
            <col name="referenceID">mycollection</col>
            <col name="value"><![CDATA[
               <root>
               <row>
                      <col name="first_name">John</col>
                      <col name="last_name">Smith</col>
               </row>
               <row>
                      <col name="first_name">Bob</col>
                      <col name="last_name">Miller</col>
               </row>
               </root>]]></col>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
            </row>
        </root>
            <error>
                <code>0</code>
                <message>No error</message>
            </error>
</result>
```

Error Handling

For HTTP and Java connectors, VoiceObjects provides a way to define error codes and messages within the connector and retrieve them later in the dialog using event handling and the two Expression functions ERRORCODE and ERRORMESSAGE. This allows you to catch errors and take appropriate actions at call time, like playing error messages to the caller or writing code and messages to a log file or database.

Note that Web Service connectors do not support this. The value of the two functions will not be changed in case of an error; only the return value will be set to the HTTP error code if an object is specified in the **Return value** field of the Connector item definition.

Error codes can be used in event handling to define specific handlers for certain error types. An event handler of type *Error – Connector* is activated if an error code above 0 (zero) is returned to the application. For more information on this, refer to Event Handling in this *Object Reference*.

The definition of the error code and message depends on the connector type.

Java connectors

If the connector is a Java program, error codes and error messages can be set using the two getter methods `public String getErrorCode()` and `public String getErrorMessage()`, respectively. The return value of getErrorCode should be a *positive* number that is associated with a certain (custom) type of error. The error code 0 (zero) and codes below 0 indicate that no error occurred. The return value of getErrorMessage is an arbitrary string.

HTTP connectors

If the connector is a CGI script, error codes and error messages can be set within an explicit error handling section in the Result XML. This section starts with the tag `<error>` and contains two sub-tags

- `<code>` defining the code for the error.
- `<message>` defining the message for the error.

The code should be a *positive* number that is associated with a certain (custom) type of error. The error code 0 (zero) and codes below 0 indicate that no error occurred. The message is an arbitrary string.



Caution: When using the HTTP connector, make sure that the back-end system that is being called does **not** close the socket being used for the request while leaving the HTTP session itself open. This can lead to subsequent problems since open HTTP sessions are pooled for re-use. When closing the socket, always close the HTTP session, too.

Copyright © 2001-2011 Voxeo Germany GmbH

HTTP Connector Framework

To ease development of Java Servlet-based back-end connectivity via a HTTP Connector, VoiceObjects provides the HTTP Connector Framework that provides an easy way to implement Java-based back-end connectivity without having to deal with the details of Servlets and JSPs.

The HTTP Connector Framework is located in the *Platform\Resources\CGIConnector* directory of your VoiceObjects installation. This directory contains a Web application that can be deployed to a Java Web application server. It includes the source code of two sample connectors that can be found in the subdirectory *\Sources\SampleConnector* in the *CGIConnector* directory.

To implement the Java part of a new CGI Connector, a Java class has to be implemented that extends the class `com.voiceobjects.connector.util.ConnectorServlet`. The new class needs to implement the method:

```
public ConnectorResult perform(HttpServletRequest servletRequest,
Map<String, ConnectorParameter> parameterMap)
```

This method is expecting two parameters – the original Servlet request that is received by the Web application server and a map containing the parameters sent by the VoiceObjects Connector object. In this map, reference IDs of the parameters are mapped to the actual parameters (instances of `ConnectorParameter`). This allows easy access to the parameters.

The method returns a `ConnectorResult` containing the return parameters for the VoiceObjects Connector object, the error code, and the error message.

A JavaDoc documentation of the framework classes can be found in the subdirectory \doc in the *CGIConnector* directory.

The following code is an excerpt of the *GetSingleStockQuote* example implementation. The full source code can be found in the *\Sources\SampleConnector* directory.

Copyright © 2001-2011 Voxeo Germany GmbH

```
24      @Override
25      public ConnectorResult perform(HttpServletRequest servletRequest,
26              Map<String, ConnectorParameter> parameterMap) {
27
28          // Check if stock name is part of input parameters
29          if (!parameterMap.containsKey("stockName")) {
30              //if not, set an appropriate error code and error message and return
31              return new ConnectorResult(4000,
32                      "Input parameter with reference ID 'stockName' is missing.");
33          }
34
35          // Check if stockName is of type Collection
36          if (parameterMap.get("stockName").getType() == ConnectorParameter.TYPE_COLLECTION) {
37              //if so, set an appropriate error code and error message and return
38              return new ConnectorResult(4001,
39                      "Input parameter with reference ID 'stockName' must not be a Collection.");
40          }
41
42          //Get input parameter from parameter map
43          String stockName = parameterMap.get("stockName").getValue();
44
45          //retrieve stock price from back-end systems
46          String stockPrice = null;
47          try {
48               stockPrice = "12.34";
49          }
50          catch (Exception e) {
51              //if an exception occurred during back-end access,
52              //set an appropriate error code and error message and return
53              return new ConnectorResult(5000, "Error accessing backend.");
54          }
55
56          //build ConnectorResult object (default error code is 0)
57          ConnectorResult result = new ConnectorResult();
58          //add the result value to the result object
59          result.addResultValue("stockPrice", stockPrice);
60
61          return result;
62      }
```

The example shows the implementation of the `perform` method for a connector that is using two parameters – `stockName` and `stockValue`. The example shows how the framework classes can be used to get the connector parameters, check the types of the parameters, and to assemble the result values.

The *GCIConnector* directory also contains the source code of another example (*GetMultipleStockQuotes*) using collections as input and output parameters.

If the *CGIConnector* directory is deployed to your Java Web application server, the sample connectors are available at *http://server:port/CGIConnector/GetSingleStockQuote* and *http://server:port/CGIConnector/GetMultipleStockQuotes*

“server” and “port” need to be substituted by the appropriate values for your application server.

To implement the HTTP server side for a CGI Connector using the CGI Connector Framework, the following steps are required:

1. Implement a Java class that extends `com.voiceobjects.connector.util.ConnectorServlet`. As shown above, this class receives the connector parameters and connects to your back-end system.
2. Compile the class and put the class files into the subdirectory *\WEB-INF\classes* in the *CGIConnector* directory. Alternatively you can create a jar file and put it into the subdirectory *\WEB-INF\lib*.
3. Edit the file *web.xml* in the subdirectory \WEB-INF in the *CGIConnector* directory. Add a `servlet` and a `servlet-mapping` section for your new class (see below for an example).
4. Deploy the *CGIConnector* directory to your Java Web application server.

Copyright © 2001-2011 Voxeo Germany GmbH

The following example shows the *web.xml* file for the sample connector implementations.

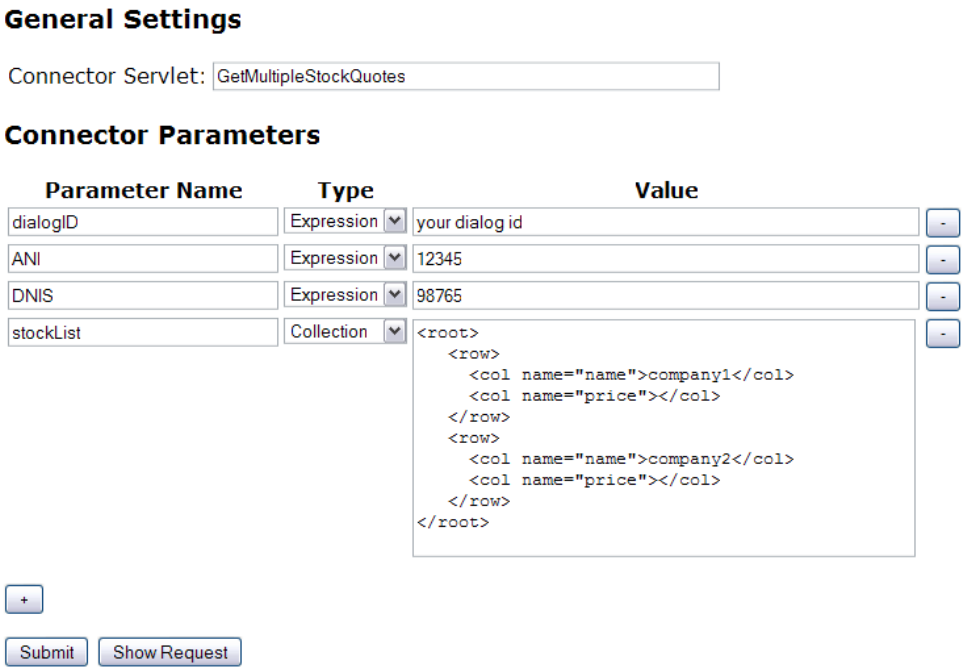
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
    <display-name>CGIConnector</display-name>
    <servlet>
        <servlet-name>GetSingleStockQuote</servlet-name>
        <servlet-class>com.voiceobjects.connector.test.GetSingleStockQuote</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet>
        <servlet-name>GetMultipleStockQuotes</servlet-name>
        <servlet-class>com.voiceobjects.connector.test.GetMultipleStockQuotes</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>GetSingleStockQuote</servlet-name>
        <url-pattern>/GetSingleStockQuote</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>GetMultipleStockQuotes</servlet-name>
        <url-pattern>/GetMultipleStockQuotes</url-pattern>
    </servlet-mapping>
</web-app>
```

Note: The CGI Connector framework requires version 6 of the Sun Java Runtime Environment.

To test the HTTP server side for a CGI Connector, the CGI Connector Framework contains the *VoiceObjects Backend Launcher*, which is a HTML testing page. The file is called *test.html* and is located in the *CGIConnector* directory. If the *CGIConnector* directory is deployed to a Java Web application server, the test page can be reached at *http://server:port/CGIConnector/test.html*.

The *Backend Launcher* allows specifying the path of the Servlet to be tested and a set of parameters that will be sent to the Servlet. For each parameter, name, type, and value can be specified. The following example shows how the *Backend Launcher* can be used to call the *GetMultipleStockQuotes* sample implementation.

Copyright © 2001-2011 Voxeo Germany GmbH



By clicking the plus button (+) you can add additional parameters; clicking the minus button (-) removes them. **Show Request** shows the request XML structure that will be sent to the Servlet. **Submit** sends the actual request.

Based on the sample implementation the request that is shown in the example above will return the following result:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
 <root>

  <row>
   <col name="referenceID">stockList</col>
   <col name="value"><![CDATA[<root><row><col name="name">company1</col><col name="price">12.34</col></row>
  <row><col name="name">company2</col><col name="price">12.34</col></row></root>]]></col>
  </row>

 </root>
 <error>

  <code>0</code>
  <message></message>
 </error>
</result>
```

Note: The *Backend Launcher* has only been tested with the *Mozilla Firefox* browser.

VoiceObjectsXML Definition

The Connector object is represented by the VoiceObjectsXML element `<connector>`. It has two attributes and eight groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<connector>` element uses the embedded **`<connectorItem>`**, **`<parameterSet>`** and `<resultMapping>` elements.

Copyright © 2001-2011 Voxeo Germany GmbH

Connector

Attributes

- **`user`**
  User ID used for optional HTTP authentication in CGI and Web Service connectors. If authentication fails, a corresponding HTTP return code will be made available through the `returnValue` of the Connector item definition. At the same time, an *Error – Connector* event is activated.
- **`password`**
  The password for the user ID provided in `user`.
- **`certificate`**
  Certificate to be used when working with SSL.
- **`mutualAuthentication`**
  Defines whether to use mutual authentication. Legal values are *true* or *false*. If not defined, defaults to *false*.
- **`clientCertificate`**
  Client certificate to be used when working with mutual authentication.
- **`clientCertificateAlias`**
  Client certificate alias to be used when working with mutual authentication.
- **`clientCertificatePassword`**
  Password for client certificate when working with mutual authentication.
- **`clientKeystore`**
  Client keystore when working with mutual authentication.
- **`clientKeystorePassword`**
  Password for client keystore when working with mutual authentication.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Connector object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Connector object.
- **`+<connectorItem>`**
  Defines the list of Connector items.
- **`<output>`**
  Defines the "process notification" output for the Connector object.
- **`<audio>`**
  Defines the wait loop audio. Alternative text for this audio is ignored.
- **`<eventHandling>`**
  Defines the event handling for the Connector object.
- **`<tuning>`**
  Defines the tuning settings for the Connector object.
- **`<sequence usage="postprocessing">`**
  Defines the postprocessing sequence for the Connector object.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<connector>
  <connectorItem location="#Connector locator"
  file="SampleConnector.jar" class="connector.PrimeInsurance"
  method="retrieveInsuranceFee">
    <parameterSet>
      <item alias="carMake" object="#Car manufacturer"/>
      <item alias="carModel" object="#Car model"/>
      <item alias="fee" object="#Insurance premium"/>
    </parameterSet>
  </connectorItem>
</connector>
```

ConnectorItem

Attributes

- **`label`**
  A text string providing a name for the Connector item.
- **`layer`**
  Defines the layer for the Connector item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **`location`**
  Defines the location from where the CGI or JAR file is to be retrieved. Must be a reference to a Resource Locator object.
  For Web Service connectors, it defines the URL of the service provider node, in case it is different from the location of the WSDL file. Typically, the location of the service provider node is the same as that of the WSDL file, in which case you can leave this attribute empty.
- **`file`**
  The file name of the CGI or JAR file. Can be a constant name, or a reference to a Variable, Expression, or Script object.
  For Web Service connectors, it is the name of the service provider node, in case it is different from the name of the WSDL file. Typically, the location of the service provider node is the same as that of the WSDL file, in which case you can leave this attribute empty.
- **`class`**
  For Java connectors, the class name within the JAR file.
  For Web Service connectors, it defines the name of the port to be used. If the WSDL file provides only one port, this attribute can be left empty.
- **`method`**
  For Java connectors, the method name within the class.
  For Web Service connectors, it defines the Web service method to be called. If there is more than one method with the same name available at the service endpoint, these methods are distinguished by the name of the input and output message. If this is the case, append the names of the input and output message to the `method` attribute, using colons as separators.

Copyright © 2001-2011 Voxeo Germany GmbH

- **`wsdlLocation`**
  Defines the URL of the WSDL file that describes the available Web services of the service provider. Must be a reference to a Resource Locator object. Only relevant for Web Service connectors.

- **`wsdlFile`**
  Defines the name of the WSDL file. Only relevant for Web Service connectors.

- **`persist`**
  Defines whether the class should be persisted, and if so at which level. Only relevant when using a Java connector. Legal values are *none*, *dialog*, *service*, *server*. If not specified, defaults to *none*.

- **`timeout`**
  For CGI and Web Service connectors, it specifies the time after which the request is aborted if no answer is received from the back-end. If the connector takes longer to process than the time specified in this field, an *Error – Connector* event is activated. If you leave it at *Default*, the setting from the corresponding Service object definition is taken.

- **`returnValue`**
  For CGI connectors, the return value is the HTTP return code received from the Web server that took the request. It can be 200 for successful requests, 404 for "File not found" exceptions, or any other HTTP return code. Note that the return value is set even if the back-end access did not succeed, e.g. in case of a 404 error.
  For Java connectors, it is the return value of the method called. If the return type of the method is *void*, the return value is empty and the linked object will not be overwritten.
  For Web Service connectors, it is the content of the response. Variable objects are typically used to store the result. Apply the *XPath* function of the Expression object to examine the result, if desired.
  Must be a reference to a Variable or Collection object.

Children

- **`<parameterSet>`**
  Defines the parameter set for the Connector item.

- **`<resultMapping>`**
  Defines the result mapping for the Connector item.

Example

```
<connectorItem location=”#CGI Locator” file=”get_data.pl”/>
  <parameterSet>
    <item object=”#Data set”/>
  </parameterSet>
</connectorItem>


<connectorItem location=”#JAR Locator” file=”dbConnect.jar”
class=”customerData” method=”getProfile” persist=”dialog”>
  <parameterSet>
    <item alias=”callerProfile” object=”#Caller profile”/>
  </parameterSet>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
</connectorItem>
```

ParameterSet

Children

- **+<item>**
  Defines the list of entries in the parameter set. The use of the `alias` attribute is optional when the `object` attribute is used, and required when the `value` attribute is used.

Example

```
<parameterSet>
  <item alias=”customer” object=”#Customer number”/>
  <item alias=”locale” value=”de-DE”/>
</parameterSet>
```

Item

Attributes

- **object**
  Defines a reference to an object. If this attribute is defined, the `value` attribute must not be defined.
- **value**
  Defines a constant value. If this attribute is defined, the `object` attribute must not be defined.
- **alias**
  Defines the parameter alias under which the parameter is accessible in the external code.

Example

```
<item alias=”premium” object=”#Annual premium”/>


<item alias=”locale” value=”en-US”/>
```

ResultMapping

Children

- **+<resultMappingItem>**
  Defines the list of entries in the result mapping. Both the `object` and `value` attribute must be defined.

Example

```
<resultMapping>
  <resultMappingItem object="#Tradevalue"
value="/quote/lastTradeValue"/>
  <resultMappingItem object="#PrevTradeValue"
value="/quote/previousCloseValue"/>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
</resultMapping>
```

ResultMappingItem

Attributes

- **`object`**
  Defines a reference to a Variable, Collection or Layer object.
- **`value`**
  Defines the XPath expression that will be applied to the return value. The result of this expression will be assigned to the object under `object`.

Example

```
  <resultMappingItem object="#Tradevalue"
value="/quote/lastTradeValue"/>

  <resultMappingItem object="#PrevTradeValue"
value="/quote/previousCloseValue"/>
```

Object Interoperability

The following table contains all object types that can refer to a Connector object:

Icon	Object Name	Use Case Example
	Module	A Connector object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, a Connector object is used within a Sequence object in a dialog flow.
	Menu	A Connector object can be used as the destination object within a Menu item in a Menu object.
	Confirmation	A Connector object can be used as the destination object within a Correction item inside a Confirmation object. While technically possible, this is not typically useful.
	If	A Connector object can be used within either the THEN or the ELSE item of an If object.
	Case	A Connector object can be used in any WHEN item of a Case object.
	Loop	A Connector object can be used within the embedded sequence of a Loop object.
	Goto	A Connector object can be referenced via a Goto object.
	Hyperlink	A Connector object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explaining object names and descriptions.

Copyright © 2001-2011 Voxeo Germany GmbH

The name of a Connector object should indicate the task it performs. The short description should contain a brief explanation of the data exchanged, as well as assumptions and restrictions. The long description should give a full description of the parameters exchanged in the Connector object, indicating whether they are read, write, or read/write. In case of finite value sets, these should also be listed (e.g. *Variable can have the values yes, no, or maybe*). The table below lists three examples:

Name	Description
Retrieve caller profile	Retrieves caller profile from database based on ANI. The profile is provided in the Collection object *Caller Profile*.
Obtain flight options	Retrieves list of possible flight connections from back-end, based on date, time, origin, and destination. Note that the list may be empty.
Store caller preferences	Writes the modified caller preferences (persona, output style, barge-in setting) to the database.

Copyright © 2001-2011 Voxeo Germany GmbH

Database

Overview

The Database object enables easy access to Relational Database Management Systems (RDBMS) to retrieve, store, and manipulate data. It can work with single values as well as entire tables, which are represented as Collection objects. The result mapping capability can be used to conveniently extract individual values from result sets.

The Database object also provides embedded connection pooling to optimize throughput and at the same time limit the number of simultaneous RDBMS connections.

Usage examples for the Database object are:

- Retrieving dynamic content such as prompts or grammars
- Retrieving customer profiles at the beginning of the call and writing back updates after the call has ended
- Making a transaction on a back-end system, e.g. booking a flight

While the Database object specializes on access to RDBMS, the Connector object is available for general purpose backend connectivity. The advantage of the Database object is that only the SQL needs to be provided; when working with the Connector object some level of custom coding is required.

The Object Definition below covers the configuration of the Database object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Database object belongs to the object category Resources.

Dialog Flow Scenario

The following dialog flow shows a database transaction while the caller is logging into a brokerage service. The personal stock portfolio of the caller is retrieved and played back as a list.

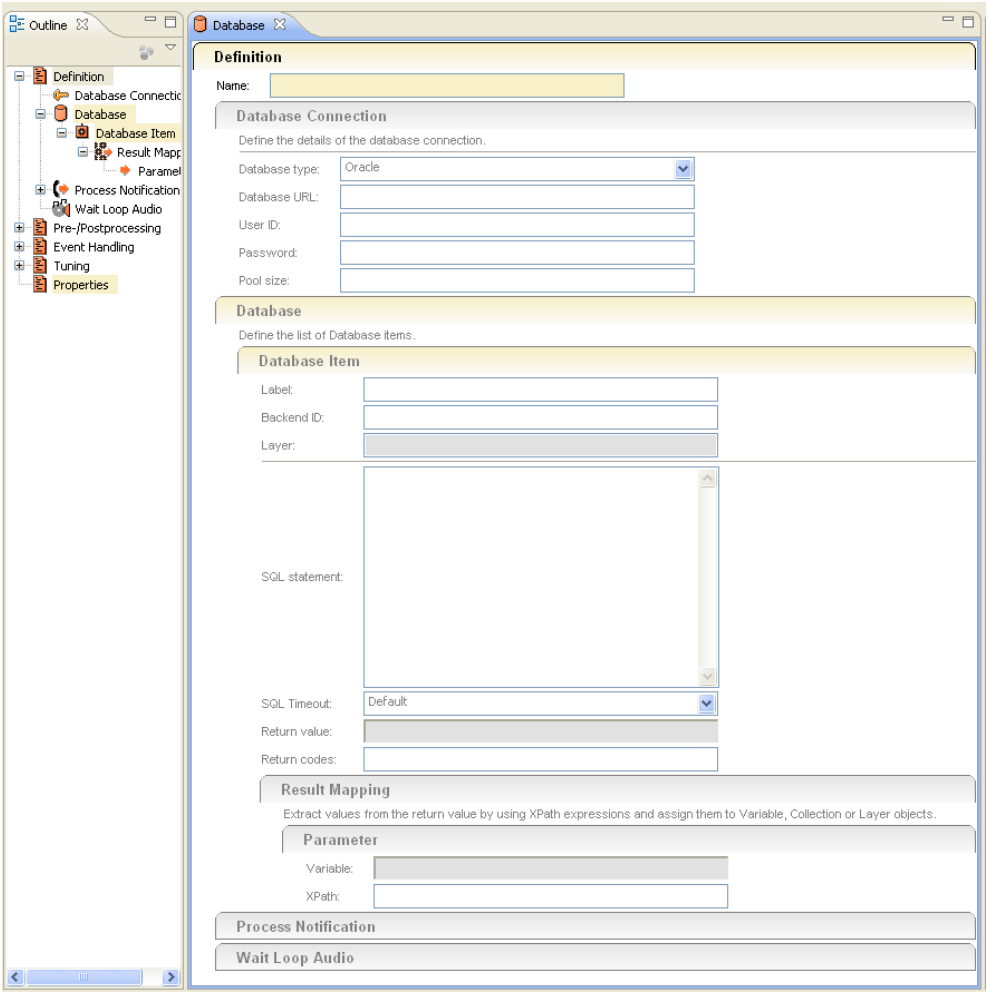
Object – Caller		Dialog Flow
		*Please wait while I check your data.*
		A wait loop audio file is played until the Database object has finished processing.
		Connects to the database holding customer data.
		[*Earcon for positive confirmation*]
		*Welcome to the brokerage service. Here's a list of all the companies in your portfolio…*

Copyright © 2001-2011 Voxeo Germany GmbH

Object Definition

The **Definition** of the Database object provides the following sections:

- Database Connection
  To define the connection to the RDBMS.
- Database
  To define one or more Database items.
- Process Notification
  To optionally specify a notification to the caller that the database processing has started.
- Wait Loop Audio
  To optionally specify an audio file that is played back while the database process is running.

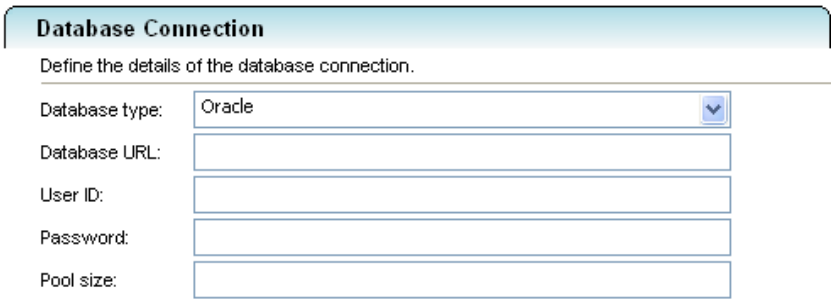


For further details regarding additional object configuration, refer to *Pre-/Postprocessing*, *Event Handling*, *Tuning*, or *Properties* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Database Connection

The **Database Connection** section defines the key properties required for connecting to the RDBMS.



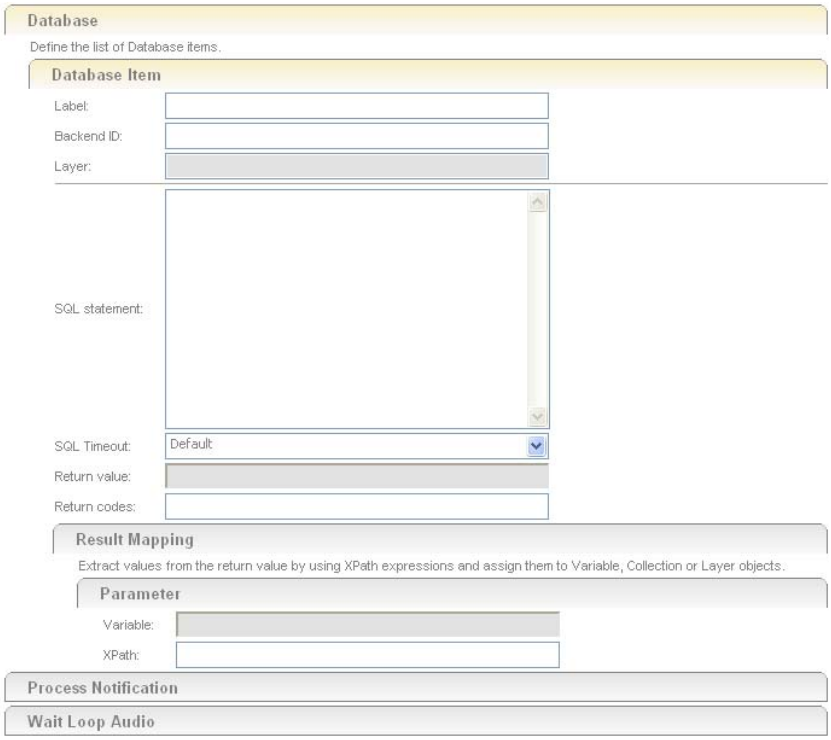
The individual properties are as follows:

Property	Description
Database type	The type of RDBMS to connect to. Predefined types are *DB2*, *Derby*, *H2*, *MaxDB*, *MySQL*, *Oracle* (the default), *PostgreSQL*, and *SQL Server*. In addition it is also possible to provide the fully qualified name of the JDBC driver when using a RDBMS not contained in this list.
Database URL	The URL to connect to the RDBMS. This is a JDBC URL e.g. of the form *jdbc:oracle:thin:@localhost:1521:OVAPDB* (for Oracle) or *jdbc:jtds:sqlserver://localhost:1433/TDR* (for SQL Server). It is also possible to utilize database connection pools configured at the level of the Web Application Server. This is done using a URL with the prefix *jndi://* such as *jndi://java:comp/env/jdbc/TestDB.* For more details refer to Using Application Server Database Pooling.
User ID	The user ID with which to connect to the RDBMS. Ignored when a jndi:// **Database URL** is used.
Password	The password for the user specified by **User ID**. Note that **User ID** and **Password** can also be defined using e.g. Variable objects that are filled by an external configuration using Application Defaults. See the corresponding section in Chapter 4 – *Service Deployment* in the *Deployment Guide* for details. Ignored when a jndi:// **Database URL** is used.
Pool size	The size of the connection pool to be used. This limits the total number of simultaneous connections open at the same time for the same RDBMS connection. This connection is defined by **Database type**, **Database URL** and **User ID**. At the same time it optimizes throughput by eliminating the overhead of constantly having to re-establish the connection. If not specified, a default pool size of 10 is used. Ignored when a jndi:// **Database URL** is used.

Copyright © 2001-2011 Voxeo Germany GmbH

Database

In the **Database** section, define at least one **Database Item** that will process a SQL statement.



Each **Database Item** contains the following set of fields:

Property	Description
Label	Optional parameter to identify the Database item in a list.
Layer	Optional parameter to indicate the layer that the Database item belongs to. For more information on layers, see Chapter 7 – *How to Use Layers* in the *Design Guide* or the Layer object in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
SQL Statement	The **SQL Statement** can be defined by entering plain text in the text area or by inserting new or existing Variable, Collection, Expression, Script, and Layer objects. For more information on how to do this, refer to Chapter 6 – *Object Editors* in the *Desktop for Eclipse Guide* and *Desktop for Web Guide*, respectively. To insert a new object, drag the appropriate object type from the **Object Palette**. In Desktop for Web this is done via the context menu. To simplify the mapping to the tables of the Infostore Repository you can use wildcards inside a SQL statement that represent key values from tables, like dialog ID, CRM ID, the ID of the active business task, etc. For a list of these wildcards, refer to Chapter 1 – *Welcome to Infostore* in the *Infostore Guide*. **Note:** Stored procedures are currently not supported. For more information on how to invoke stored procedure from VoiceObjects, refer to *http://evolution.voxeo.com/wiki/kb:storedProcedures*.
SQL Timeout	Specifies the time after which the request is aborted if no answer is received from the database. If the processing of the SQL takes longer to process than the time specified in this field, an *Error – Connector* event is activated. You can select one of the values of the drop-down list or enter a custom value. Custom values are interpreted as seconds. If you leave the value at *Default*, the default HTTP Connector timeout setting from the corresponding Service object definition is taken.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description

Return value	Allows to link a Variable or Collection object that will be set to the return value of the SQL statement.
	The structure of the return value depends on whether the SQL statement selects values from tables, or modifies tables (e.g. using UPDATE or INSERT).
	When values are selected, the structure of the return value is as follows:
	• If a single value is returned by the RDBMS and a Variable object is linked as return value, the variable receives the single value.
	• If a single value is returned by the RDBMS and a Collection object is linked as return value, the collection receives a structure of the form
	`<root>`
	`  <row>`
	`  <column name="Column name from database table">Value</column>`
	`  </row>`
	`</root>`
	• If a set of rows is returned by the RDBMS and a Variable object is linked as return value, the variable receives the comma-separated list of values, column-by-column and row-by-row.
	• If a set of rows is returned by the RDBMS and a Collection object is linked as return value, the collection receives a structure of the form
	`<root>`
	`  <row>`
	`  <column name="Column name from database table">Value</column>`
	`  </row>`
	`  …`
	`</root>`
	
	When tables are modified, the structure of the return value is as follows:
	• If a Variable object is linked as return value, the variable receives the number of rows that were modified.
	• If a Collection object is linked as return value, the collection receives a structure of the form
	`<root>`
	`  <row>`
	`  <column name="cnt">Number of modified rows</column>`
	`  </row>`
	`</root>`

Result Mapping

The **Result Mapping** section allows extracting specific values from the return value of the SQL statements by applying an XPath expression to the return value. Any number of result mappings can be defined. The expression specified in the **XPath** field will be applied to the return value of the SQL statement. The result of this operation will be assigned to the Variable, Collection or Layer object defined in the **Variable** field.

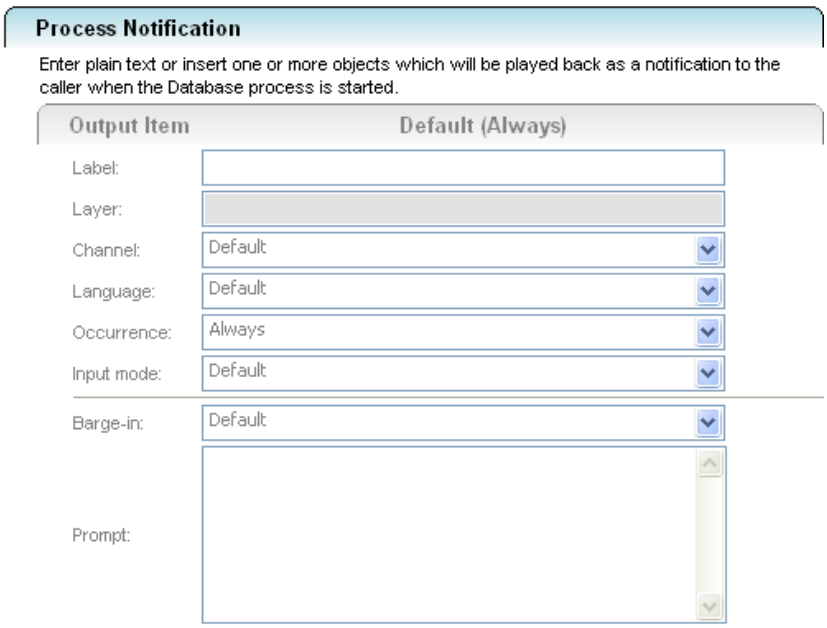
Copyright © 2001-2011 Voxeo Germany GmbH

For more information on using **Result Mapping**, refer to the Connector object.

More information about XPath can be found at http://www.w3.org/TR/xpath.

Process Notification

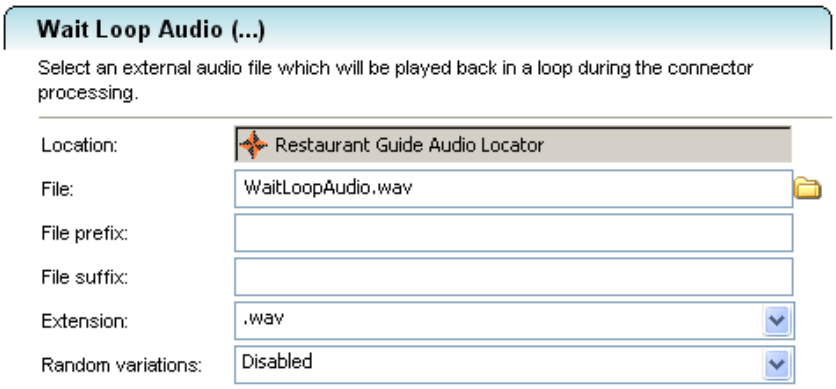
In the **Process Notification** section, an embedded Output object can be specified that gives the caller a notification that the database process is starting. This parameter is optional.



For further details, refer to the Output object in this *Object Reference*.

Wait Loop Audio

In the **Wait Loop Audio** section, an audio file can be specified that is played back while the database process is running. This parameter is optional and only applies to the voice and video channel. It is typically used for time-consuming database operations, e.g. to play hold music.



Copyright © 2001-2011 Voxeo Germany GmbH

For further details on the possible settings, refer to the Audio object in this *Object Reference*.

Note: On some media platforms, the duration of the audio file must be at least as long as the processing time needed by the Database object. On other platforms, an audio file that is shorter than the processing time required by the database will be looped continuously as needed. For further information on the behavior of audio processing refer to the local documentation of the media platform vendor.

Processing the Database Object

This section contains some technical remarks on the treatment of Database objects in VoiceObjects Server.

For each Database object used in the voice or video channel, the server sends a VoiceXML document (a single one for all Database items) to the media platform, even though there is no interaction with the caller. The document consists of a `<submit>` element including the *fetchaudio* attribute. If a wait loop audio is specified for the object, the audio is mentioned in this attribute, so that the platform plays this file while the connector is processed. Any output defined before the Database object (either in the **Process Notification** section within the object, or in Output objects prior to the Database object in a sequence) will be "flushed" by the platform, i.e. played back before the wait loop audio is applied, which is a desirable effect. To achieve this effect even if no wait loop audio is defined, the server renders a reference to an empty audio file (taken from the system resource folder of the VoiceObjects installation) in the *fetchaudio* attribute. Without this "flush" effect, a media platform might withhold the output defined prior to the Database object until it has been processed, which would result in undesirable delays.

At the same time, a *fetchtimeout* attribute is rendered and set to the value of *10000s* (ten thousand seconds). The effect of this is that the media platform is forced to wait until the Database object has been processed and does not try to request another VoiceXML document from the server before then. If it did, the server would reject the request.

In the text and Web channel, there will be no intermediate page rendered, unless there is a process notification defined which is rendered as a standalone output.

The Database object waits for the result of the **SQL Statement** defined in each Database item until the respective **SQL Timeout** expires, in which case an *Error – Connector* event is activated and the processing of any subsequent Database items is aborted. Note that, depending on the RDBMS, processing on the side of the database may continue even after VoiceObjects Server has aborted the Database object processing after the timeout.

Using Application Server Database Pooling

VoiceObjects Server maintains a configurable pool of database connections to preserve resources and maximize throughput. In some cases, though, it can be preferable to make use of pre-existing database pools configured on the level of the Web Application Server (such as e.g. Tomcat). This is possible by using a specially formatted **Database URL**.

The connection pool configured in the application server is looked up using JNDI, which is why the Database URL has a format such as *jndi://java:comp/env/jdbc/TestDB*. The first portion (*java:comp/env*) denotes the generic context, while the second portion (in this case *jdbc/TestDB*) specifies the name assigned in the application server configuration.

Copyright © 2001-2011 Voxeo Germany GmbH

For information on how to define the JDBC connection pool itself e.g. in Tomcat, refer to http://tomcat.apache.org/tomcat-5.5-doc/jndi-resources-howto.html#JDBC_Data_Sources.

VoiceObjectsXML Definition

The Database object is represented by the VoiceObjectsXML element `<database>`. It has five attributes and eight groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<database>` element uses the embedded **`<databaseItem>`** and `<resultMapping>` elements.

Database

Attributes

- **`user`**
  User ID used for database schema.
- **`password`**
  The password for the user ID provided in `user`.
- **`dbURL`**
  URL to connect to the RDBMS.
- **`dbType`**
  Type of RDBMS used. Either one of *DB2*, *Derby*, *H2*, *MaxDB*, *MySQL*, *Oracle*, *PostgreSQL*, *SQLServer*, or the fully qualified name of the JDBC driver for the RDBMS (e.g. com.mysql.jdbc.Driver).
  Default value is *Oracle*.
- **`poolsize`**
  Size of the pool to be used for this database connection. Must be an Integer. If not defined, defaults to 10.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Database object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Database object.
- **`+<databaseItem>`**
  Defines the list of Database items.
- **`<output>`**
  Defines the "process notification" output for the Database object.
- **`<audio>`**
  Defines the wait loop audio. Alternative text for this audio is ignored.
- **`<eventHandling>`**
  Defines the event handling for the Database object.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<tuning>**
  Defines the tuning settings for the Database object.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the Database object.

Example

```
<database name="Retrieve language preference" dbType="SQLServer"
  dbURL="prefdb1:1433/CALLERPREFS" user="prefmanager"
password="c78gbk4">
  <databaseItem returnValue="#Language Preference">
    <text>select LANG from CALLERPREFS where CALLERID='</text>
    <variable link="#Caller ID"/>
    <text>';</text>
  </databaseItem>
</database>
```

DatabaseItem

Attributes

- **label**
  A text string providing a name for the Database item.
- **layer**
  Defines the layer for the Database item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form "Layer=State" or "Layer!=State" where "State" is the label of a state for the layer "Layer".
- **timeout**
  The timeout value of the SQL statement executed on the RDBMS. Must be *default* or an Integer. If not defined, defaults to *default*.
- **returnValue**
  The return value of the SQL statement executed on the RDBMS.
  Must be a reference to a Variable or Collection object.

Children

- **+(<output>, <variable>, <expression>, <collection>, <script>, <text>, <layer>)**
  Defines the SQL statement, as a concatenation of various elements.
- **<resultMapping>**
  Defines the result mapping for the Database item.

Example

```
  <databaseItem returnValue="#Language Preference">
    <text>select LANG from CALLERPREFS where CALLERID='</text>
    <variable link="#Caller ID"/>
    <text>';</text>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
</databaseItem>
```

ResultMapping

Children

- **+<resultMappingItem>**
  Defines the list of entries in the result mapping. Both the `object` and `value` attribute must be defined.

Example

```
<resultMapping>
  <resultMappingItem object="#Tradevalue"
value="/quote/lastTradeValue"/>
  <resultMappingItem object="#PrevTradeValue"
value="/quote/previousCloseValue"/>
</resultMapping>
```

ResultMappingItem

Attributes

- **object**
  Defines a reference to a Variable, Collection or Layer object.
- **value**
  Defines the XPath expression that will be applied to the return value. The result of this expression will be assigned to the object under `object`.

Example

```
  <resultMappingItem object="#Tradevalue"
value="/quote/lastTradeValue"/>
  <resultMappingItem object="#PrevTradeValue"
value="/quote/previousCloseValue"/>
```

Object Interoperability

The following table contains all object types that can refer to a Database object:

Icon	Object Name	Use Case Example
	Module	A Database object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, a Database object is used within a Sequence object in a dialog flow.
	Menu	A Database object can be used as the destination object within a Menu item in a Menu object.
	Confirmation	A Database object can be used as the destination object within a Correction item inside a Confirmation object. While technically possible, this is not typically useful.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Object Name	Use Case Example
	If	A Database object can be used within either the THEN or the ELSE item of an If object.
	Case	A Database object can be used in any WHEN item of a Case object.
	Loop	A Database object can be used within the embedded sequence of a Loop object.
	Goto	A Database object can be referenced via a Goto object.
	Hyperlink	A Database object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explaining object names and descriptions.

The name of a Database object should indicate the task it performs. The short description should contain a brief explanation of the data exchanged, as well as assumptions and restrictions. The long description should give a full description of the parameters exchanged in the Database object, indicating whether they are read, write, or read/write. In case of finite value sets, these should also be listed (e.g. *variable can have the values yes, no, or maybe*). The table below lists three examples:

Name	Description
 Retrieve caller profile	Retrieves caller profile from database based on ANI. The profile is provided in the Collection object *Caller Profile*.
 Obtain flight options	Retrieves list of possible flight connections from back-end, based on date, time, origin, and destination. Note that the list may be empty.
 Store caller preferences	Writes the modified caller preferences (persona, output style, barge-in setting) to the database.

Copyright © 2001-2011 Voxeo Germany GmbH

Event Handling

Overview

Event handling specifies how to proceed within a dialog in the case of any exceptions. In the voice and video channel in particular, various kinds of exceptions may occur during a dialog. Standard events are, for example, No Input - the caller does not respond to a question from the application – or No Match – the utterance cannot be matched to the active grammars. The VoiceObjects platform distinguishes between server-internal and media platform-related events. Server-internal events may occur in all channels, whereas media platform-related events are specific for the voice and video channel. This means that handlers for all platform-related event types will be ignored in the text and Web channel. See Server-Internal vs. Platform Events for more information on this.

Event Levels

Typically, an application will need event handling for the standard events No Match and No Input. For each dialog step, every event can occur never, once, or several times in a row, depending on the caller's behavior. The Voice User Interface designer needs to define how to react to these events, taking the occurrence level of an event into account. For example, the designer may choose to play the prompt "*Excuse me?*" for No Match, level 1 (meaning the first occurrence of this event at the current dialog step), and then listen to the caller again. For No Match, level 2, a prompt "*I still didn't understand you*" could be played, followed by a context-sensitive message that repeats what the caller can say at that point in the dialog. For the No Input event, similar strategies could be used. If both events occur three or four times in a row at one dialog step, the designer may decide to have the application perform a transfer to a call center agent. Note that the occurrence level of an event is reset when the caller has successfully responded to a question, i.e. when the input state has been exited and the dialog proceeds.

The following dialog flow scenario shows how a voice application might deal with consecutive No Input and No Match events occurring in an input state that asks for the caller's age:

Object - Caller	Dialog Flow
	How old are you?
Caller	[does not respond within 4 seconds]
	[No Input 1:] *Please tell me your age*
Caller	*I... err... my age is twenty.*
	[No Match 1:] *Excuse me?*
Caller	[does not respond within 4 seconds]

Copyright © 2001-2011 Voxeo Germany GmbH

Object - Caller	Dialog Flow
	[No Input 2:] *Sorry, I still did not hear you. Please say your age, for example 'twenty-five', or enter it on your keypad.*
 Caller	[DTMF:] *2 0*
	*Thank you.*

For detailed information on how to build a sophisticated event handling mechanism including the reprompting functionality, see Chapter 8 – *Advanced Event Handling* in the *Design Guide*.

For further information on event handling strategies with respect to aspects of Voice User Interface design also refer to Event Handling Strategies in Appendix A - *Voice User Interface Design* in the *Design Guide*.

Inheritance

Event handling can be specified on different levels of your application. Module objects are used to specify global event handling whereas the object-specific events may be set locally on the subordinate objects themselves.

Event handling is inherited starting from the first processed Module object, down through any other Module objects used within the dialog flow, to the current object. If different event handling items are defined for the same event type and occurrence level, the lowest one (i.e. the closest when moving up from the current object) within the dialog flow will be processed.

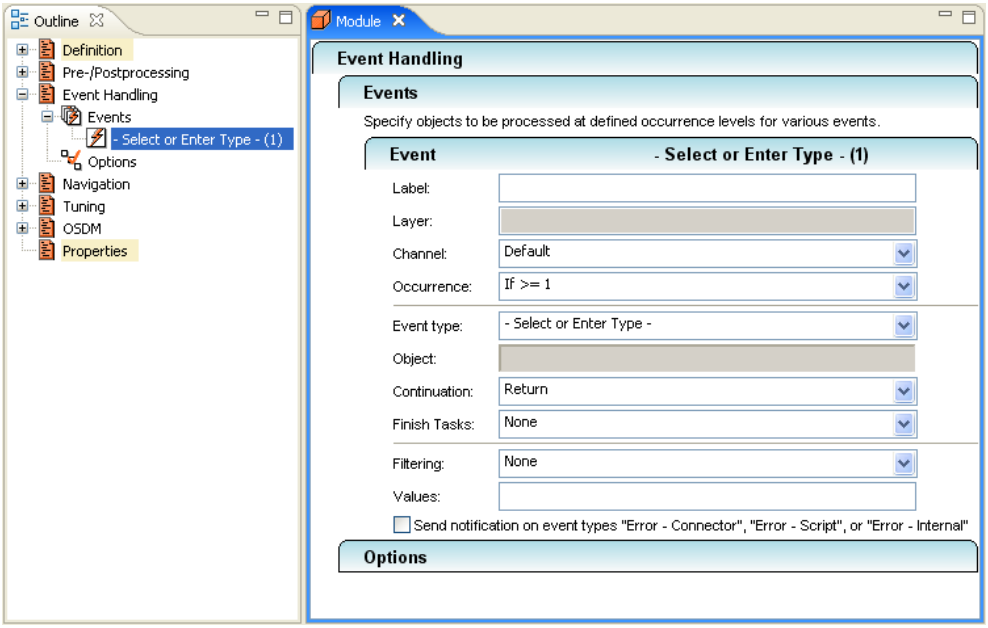
Inheritance can be disabled in the **Options** section of the **Event Handling** definition. When switched off, no event handling items are inherited into the current object but only locally defined ones (if any) are used.

Event Handling Definition

The definition of event handling described in this paragraph refers to VoiceObjects Desktop. For information on how to define event handling using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

Event Handling can be defined in the **Event Handling** section of the respective object editors as for instance the **Module** editor (entered by clicking it in the **Outline** in Desktop for Eclipse; represented by a tab in Desktop for Web).

Copyright © 2001-2011 Voxeo Germany GmbH



In the **Event Handling** section you can specify the following parameters for each event handling item:

Label
In the **Label** field, you may optionally enter a label to identify a specific event handling item more easily within a list.

Event type
In the **Event type** field, select one of the following event types:

Event	Description
ASR – Invalid Answer Limit	Counts all No Input and No Match events within a single input state and is activated if the caller reaches the defined limit of invalid answers. E.g. if an *ASR – Invalid Answer Limit* event with occurrence 3 is defined, it will be activated if the caller causes a total of three No Input or No Match events in any combination. Note that if multiple *ASR – Invalid Answer Limit* events with different occurrence levels are active, only the one with the lowest occurrence is used. Due to the way events are counted in VoiceXML, the *ASR – Invalid Answer Limit* event only works properly when the handlers for *No Input* and *No Match* remain within the same VoiceXML page, i.e. do not use Goto or Exit objects. The *ASR – Invalid Answer Limit* event does not work in combination with OSDM objects.
ASR – Max Processing Time	The caller exceeded the maximum recording or transfer time. Can only be used in Recording and Transfer objects.

Copyright © 2001-2011 Voxeo Germany GmbH

Event	Description
ASR – Max Speech Duration	The caller exceeded the maximum speech time. Can be used in all input state dialog objects except Recording (see *Max Processing Time*).
ASR – No Input	The caller did not respond to a request for input within the timeout interval.
ASR – No Input/Match	A No Input or a No Match event occurred. Using the combined handler has the same effect as defining individual No Input and No Match handlers with the same settings.
ASR – No Match	The caller said or entered something, but it did not match any of the currently active input grammars.
ASR – No Match (DTMF)	The caller used DTMF for his last input and typed in DTMF keys that did not match any of the active grammars.
ASR – No Match (Voice)	The caller used speech for his last input and said something that did not match any of the active grammars.
Caller – Cancel	The caller said the universal command *Cancel*, or a hyperlink triggered the cancel event.
Caller – Exit	The caller said the universal command *Exit*, or a hyperlink triggered the exit event.
Caller – Exit/Cancel	The caller said the universal commands *Exit* or *Cancel*, or a hyperlink triggered the exit or cancel event.
Caller – Help	The caller said the universal command *Help*, or a hyperlink triggered the help event.
Disconnect	A *Disconnect Hangup* or *Disconnect Transfer* event occurred. **Note:** The Disconnect event should not be used to trigger clean-up tasks at the end of a dialog session. This should be done using dialog end processing as described in the Module object.
Disconnect – Hangup	The caller hung up. Typically, an Exit object is used in this event to send custom markup code to the media platform.
Disconnect – Transfer	The caller ended the dialog via a blind transfer. Typically, an Exit object is used in this event to send custom markup code to the media platform.
Error – Connector	An error occurred during the processing of a Connector or Database object. Note that for this event, the occurrence level setting is ignored.

Copyright © 2001-2011 Voxeo Germany GmbH

Event	Description
Error – Internal	An internal error occurred within VoiceObjects Server. This includes errors during the processing of Expression objects (e.g. a division by zero), Script objects (e.g. invalid JavaScript code), or other internal issues. If no specific event handling has been defined for *Error Connector* (or *Error Script*), a connector or database (or script) problem will also activate this error type. Note that for this event, the occurrence level setting is ignored.
Error – Media Platform	An error occurred on the media platform. This error may be an installation or configuration issue on the media platform, missing or invalid resource files like grammars, or other invalid code encountered during processing (e.g. from a Plug-In object).
Error – Script	An error occurred during the processing of a Script object. Note that for this event, the occurrence level setting is ignored.
OSDM – Max Invalid Answers	The caller reached the maximum number of allowed invalid answers in an OSDM object.
OSDM – Max No To Confirms	The caller reached the maximum number of confirmation rejections in an OSDM object.
OSDM – Max Repeats	The caller reached the maximum number of allowed repetitions in an OSDM object.
OSDM – Max Turns	The caller reached the maximum number of allowed turns in an OSDM object.

There is a difference between events that are activated by the media platform and those that are activated by the server. See Server-Internal vs. Platform Events below for a description of this difference.

For more information on the use of input mode specific No Match handlers, *ASR – No Match (Voice)* and *ASR – No Match (DTMF)*, see Chapter 8 – *Advanced Event Handling* in the *Design Guide*.

You may also handle custom events not listed above. This can be useful to handle more specific (error) events that the media platform might activate; note that the predefined event type *Error – Internal* handles all subtypes of error events.
To do this, you can either type a custom event name into the **Event type** field, or you can define the event type dynamically by referencing a Variable, Expression, Layer, or Script object. The content of the respective object needs to be the fully qualified name of the event, e.g. “error.custom.noresource”. For a full list of error events supported in VoiceXML, see the VoiceXML specification at http://www.w3.org/TR/voicexml20/#dml5.2.6.

Object
In the **Object** field, select the object to be processed in the case of the specified event. Only Output, Goto, or Exit objects can be selected here. Other objects need to be accessed via a Goto object.

Copyright © 2001-2011 Voxeo Germany GmbH

Occurrence
In the **Occurrence** field, select the occurrence level. The occurrence level specifies how many times an event must have occurred within the same input state before the specified event handling item is processed.

Example: Specify two No Input events, one with Occurrence >=*1* and the other with Occurrence >=*3*. The first two times a No Input event occurs, the first event handling item will be processed. Starting with the third time and thereafter the second event handling item is used.

Layer
In the **Layer** field a custom layer can be defined using a Variable, Expression, Script, Layer, or Collection object to dynamically enable or disable the event handling item.

Continuation
In the **Continuation** field you can define what happens after the event handling item has been processed:

- *Return* – Return to the dialog flow expecting input from the caller again. Note, that the prompt is not replayed if an Output object is linked into the **Object** field. If you want the prompt to be replayed, link a Goto object instead, and select the **Return** check box in the definition of this Goto object.
- *Reprompt* – Return to the dialog flow and play the appropriate reprompting output before expecting another input. For further details on reprompting, refer to Chapter 8 – *Advanced Event Handling* in the *Design Guide*.
- *Proceed* – Proceed with the next object in the dialog flow. The current object is skipped.



Caution: *Reprompt* does not work together with Goto or Exit objects. When you link a Goto or Exit object in the **Object** field and select *Reprompt* in the **Continuation** field, then the Goto or Exit object is not processed. When using Goto or Exit objects, you can only select *Return* or *Proceed*.

Channel
In the **Channel** field, you can set a specific channel or a combination of channels in which the event handling item is to be active. Leave it at *Default* if you want the event handling item to be active in all channels.

Finish Tasks
Defines whether currently active business tasks are to be finished or not when the event handling item is processed. It can be set to *None* (the default), which means that no task will be finished, *Active*, which means that only the currently active task will be finished, or *All*, which means that all tasks, active or inactive, will be finished. In case one or more tasks get finished, they will automatically be logged as *incomplete* to Infostore, with a status that depends on the event type. The following table lists each possible state with its corresponding event type. The values in parentheses denote the value that is internally used, e.g. in VoiceObjectsXML or as return values of the TASKSTATUS function.

Status	Set on event types
Backend error (backEndError)	*Error – Connector*

Copyright © 2001-2011 Voxeo Germany GmbH

Status	Set on event types
Business logic (businessLogic)	None
Caller abort (callerAbort)	*Caller – Exit* *Caller – Cancel* *Caller – Help* All custom event handlers
Recognition failure (recognitionFailure)	*ASR – Invalid Answer Limit* *ASR – Max Processing Time* *ASR – Max Speech Duration* *ASR – No Input* *ASR – No Match* *ASR – No Match (Voice)* *ASR – No Match (DTMF)*
Task restart (taskRestart)	None
Session termination (sessionTermination)	*Disconnect* *Disconnect – Hangup* *Disconnect – Transfer*
Technical error (technicalError)	*Error – Internal* *Error – Media Platform* *Error – Script*

Note: Finish Tasks is only applicable if a Goto or Exit object is used in the event handling item. In case of an Output object, the **Finish Tasks** setting has no effect.

Filtering
In the case of an event handler of type *Error – Connector*, select *Error Code* from the **Filtering** drop-down list if you want to define a handler for specific error codes only. For generic handlers of type *Error – Connector* leave the **Filtering** drop-down list at *None*. See the Connector object for more information on how to return error codes to the application.

Values
The codes that activate the handler are listed in the **Values** field. Enter either a single numerical code or a comma-separated list of codes. By doing this, the corresponding handler is activated only if the error code returned from the connector is in this list of values. If you set **Filtering** to *Error Code* but leave the **Values** field empty, the handler is activated for “all other error codes” (except zero). In this case, the server first checks the event handling items with non-empty values lists to deal with the more specific handlers, and then goes over to the ones with an empty list.

The option **Send notification on event types “Error – Connector” or “Error – Internal”** allows you to send a notification for event handlers of type *Error – Connector* and *Error – Internal*. If the check box is selected and the handler is activated at call time, the server sends a notification with the ID

Copyright © 2001-2011 Voxeo Germany GmbH

1.3.6.1.4.1.24140.1.2.5.8.1 and the message *Error Internal* or *Error Connector*, respectively, depending on the event type defined.

For details on notifications, refer to Chapter 8 – *Notifications* in the *Deployment Guide*.

VoiceObjectsXML Definition

Event handling is defined using the `<eventHandling>` element.

It has one attribute `inheritance`, which indicates whether event handling from parent objects should be used (*true*, the default) or not (*false*).

In addition, the `<eventHandling>` element contains one or more `<eventHandlingItem>` elements.

The `<eventHandlingItem>` element has the following attributes:

- **`label`**
  A text string providing a name for the event handling item.
- **`layer`**
  Defines the layer for the event handling item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **`channel`**
  Defines the channel(s) for which this event handling item is valid. Can be *default*, *voice*, *video*, *text*, *web*, *voiceVideo*, or *textWeb*. If not specified, defaults to *default*.
- **`eventType`** [required]
  Defines the event type to be handled by this event handling item. One of *invalidAnswerLimit*, *noInput*, *noMatch*, *noMatchVoice*, *noMatchDTMF*, *noInputNoMatch*, *maxInvalidAnswers*, *maxNoToConfirms*, *maxRepeats*, *maxSpeechDuration*, *maxTurns*, *help*, *exit*, *cancel*, *exitCancel*, *errorMediaPlatform*, *errorInternal*, *errorConnector*, *errorScript*, *disconnect*, *disconnectHangup*, *disconnectTransfer*. In addition, custom event types may be used.
  Alternatively, a reference to an Expression, Script, or Variable object can be provided.

Event	Description
cancel	The caller said the universal command *Cancel*.
cancelExit	The caller said the universal commands *Exit* or *Cancel*.
disconnect	A *Disconnect Hangup* or *Disconnect Transfer* event occurred.
disconnectHangup	The caller hung up. Typically, an Exit object is used in this event to send custom markup code to the media platform.
disconnectTransfer	The caller ended the dialog via a blind transfer. Typically, an Exit object is used in this event to send custom markup code to the media platform.

Copyright © 2001-2011 Voxeo Germany GmbH

Event	Description
errorConnector	An error occurred during the processing of a Connector or Database object. Note that for this event the occurrence level setting is ignored.
errorInternal	An internal error occurred within VoiceObjects Server. This includes errors during the processing of Expression objects (e.g. a division by zero), Script objects (e.g. invalid JavaScript code), or other internal issues. If no specific event handling has been defined for *Error Connector* (or *Error Script*), a connector or database (or script) problem will also activate this error type. Note that for this event the occurrence level setting is ignored.
errorScript	An error occurred during the processing of a Script object. Note that for this event the occurrence level setting is ignored.
errorMediaPlatform	An error occurred on the media platform. This error may be an installation or configuration issue on the media platform, missing or invalid resource files like grammars, or other invalid code encountered during processing (e.g. from a Plug-In object).
exit	The caller said the universal command *Exit*.
help	The caller said the universal command *Help*.
invalidAnswerLimit	Counts all No Input and No Match events within a single input state and is activated if the caller reaches the defined limit of invalid answers. E.g. if an *ASR – Invalid Answer Limit* event with occurrence 3 is defined, it will be activated if the caller causes a total of three No Input or No Match events in any combination. Note that if multiple *ASR – Invalid Answer Limit* events with different occurrence levels are active, only the one with the lowest occurrence is used. Due to the way events are counted in VoiceXML, the *ASR – Invalid Answer Limit* event only works properly when the handlers for *No Input* and *No Match* remain within the same VoiceXML page, i.e. do not use Goto or Exit objects. The *ASR – Invalid Answer Limit* event does not work in combination with OSDM objects.
maxNoToConfirms	The user has reached the maximum number of “no” answers to a confirmation request. This event is only relevant when using OSDMs.
maxRepeats	The user has reached the maximum number of repetitions of the same input state. This event is only relevant when using OSDMs.
maxSpeechDuration	The caller has exceeded the maximum speech time.

Copyright © 2001-2011 Voxeo Germany GmbH

Event	Description
maxTurns	The user has reached the maximum number of total turns. This event is only relevant when using OSDMs.
noInput	The caller has not responded to a request for input within the timeout interval.
noInputNoMatch	A No Input or a No Match event occurred. Using the combined handler has the same effect as defining individual No Input and No Match handlers with the same settings.
noMatch	The caller said or entered something, but it did not match any of the currently active input grammars.
noMatchDTMF	The caller used DTMF for his last input and typed in DTMF keys that did not match any of the active grammars.
noMatchVoice	The caller used speech for his last input and said something that did not match any of the active grammars.

- **`occurrence`** [required]
  Defines the occurrence level to be handled by this event handling item. Must be an integer between 1 and 10.
- **`continuation`**
  Defines the continuation mode. Legal values are *return*, *reprompt*, *proceed*. If not specified, defaults to *return*.
- **`errorCodeFilter`**
  Defines whether this event handler should use error code filtering. Only relevant for event types *errorConnector* and *errorScript*. Either *true* or *false*. If not specified, defaults to *false*.
- **`errorCodeList`**
  Comma-separated list of error codes for which this event handler applies. Only relevant for event types *errorConnector* and *errorScript*, and if *errorCodeFilter* is *true*. The list of error codes can be defined statically, or via a reference to a Variable, Expression, or Script.
- **`sendNotification`**
  Indicates whether to send a notification when this event handler is activated. Only relevant for event types *errorConnector*, *errorScript*, and *errorInternal*. Either *true* or *false*. If not specified, defaults to *false*.
- **`finishTasks`**
  Defines whether business tasks are to be finished or not. One of *none*, *active*, *all*. If not specified, defaults to *none*.

As children, the `<eventHandlingItem>` element uses `<output>`, `<goto>`, or `<exit>`.

Server-Internal vs. Platform Events

The **Event type** defined for an event handling item covers events that are activated by the media platform, and events that can only be activated internally by VoiceObjects

Copyright © 2001-2011 Voxeo Germany GmbH

Server. The latter are *Error – Internal*, *Error – Connector* and *Error – Script*. They are activated if a general error occurs while processing a dialog object (*Error – Internal*), a Java, CGI or Web Service connector (*Error – Connector*), or script code (*Error – Script*). All four channels support these event types.

All remaining types are platform-initiated events and only supported in the voice and video channel. They will be ignored in the text and Web channel. They are activated in voice or video applications if something "irregular" happens during the interaction between media platform and caller (e.g. *ASR - No Match* or *Disconnect – Hangup*).

The event handlers that handle the platform events are always rendered into the VoiceXML code, so that the media platform knows how to deal with this kind of events. This includes the handler for *Error – Media Platform*. The internal events, on the other hand, are handled completely within the server, i.e. while the server is preparing the next dialog step and before sending out a VoiceXML document to the platform. All internal events have the restriction that their occurrence level setting is ignored; the server will simply take the handler that is defined first in the "closest" corresponding object (e.g. Connector, Script, or Module), i.e. taking the inheritance hierarchy into account.

ACTIVATEEVENT function

The Expression function ACTIVATEEVENT allows activating custom events by using an Expression object at some point in the dialog flow. The event handlers for these events are also processed internally, even if the event name is that of a platform event (like *noMatch*). The reason for this is that the Expression object is never rendered as a single dialog step (even though the event handlers for these events are rendered in the voice and video channel into the VoiceXML documents of dialog steps that inherit the handlers).
If you want to activate a handler for a platform event using the ACTIVATEEVENT function, use one of the following values:

noInput, *noMatch*, *noMatchVoice*, *noMatchDTMF*, *noInputNoMatch*, *help*, *maxInvalidAnswers*, *maxNoToConfirms*, *maxProcessingTime*, *maxRepeats*, *maxSpeechDuration*, *maxTurns*, *exit*, *cancel*, *exitCancel*, *errorMediaPlatform*, *errorInternal*, *errorConnector*, *errorScript*, *disconnect*, *disconnectHangup*, *disconnectTransfer*

As the events activated with this function are processed internally, the handlers will be supported in all four channels.

The processing of this function works as follows:

When an event is activated through this function, the server checks whether a corresponding handler is found at the current point in the event handling inheritance structure. If that is the case, the target object is processed. Note that the continuation modes *Reprompt* and *Return* are treated as *Proceed*, since the event is never triggered in the immediate scope of an input state object, but in an Expression object instead. Therefore, processing continues with the next object in the flow after the Expression object, once the target object has been finished (given that the second argument of ACTIVATEEVENT, "return", is set to *true*). If there is a handler and it does not have a target object, but only a continuation mode (like *Proceed*, or *Reprompt*), ACTIVATEEVENT is ignored and processing continues with the next object (again only if "return" = *true*).
Note also that the occurrence level of the handlers for this event will be ignored as well.
When the event name uses the dot notation (e.g. *myevent.flavour1.subflavour2*), the search for corresponding handlers will include more general handlers. The server starts by looking at the most local module scope and tries to find a handler that exactly matches the event name. If none can be found, it stays in the current level and starts

Copyright © 2001-2011 Voxeo Germany GmbH

cutting qualifiers off the event name from the right and continues searching. For the above example this means that the server first looks for a *myevent.flavour1.subflavour2* handler, then for *myevent.flavour1*, then for *myevent*. If none can be found, the server climbs up one step in the module hierarchy and repeats the search.

Note: As a result, a more generic handler at a level closer to the current object will have precedence over an exact handler that's further up in the hierarchy.

If no suitable handler can be found, an *Error – Internal* event is activated and an ERROR message is written to the log files.

Copyright © 2001-2011 Voxeo Germany GmbH

Exit

Overview

The Exit object terminates a session on the server, driven by the application logic or the caller. When a session ends, the caller is either disconnected from the media platform or handed back to a higher-level application, depending on the *exit type*. The Exit object is often used within a Hyperlink object, allowing the caller to end an application using a predefined command like *good-bye*. The designer of the application can also decide to use the Exit object to end the call after a certain number of No Input or No Match events.

The Object Definition below covers the configuration of the Exit object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Exit object belongs to the object category Actions.

Dialog Flow Scenario

The first dialog flow example describes a standard exit dialog scenario within a voice portal in which the Exit object is used as an exit option within the main menu.

Object – Caller	Dialog Flow
Module	[Music jingle] [*Hello and welcome to the Voice Portal.*]
Menu	*Main menu! You have the following options … If you would like to leave the Voice Portal just say exit.*
Caller	*Exit*
Exit	*Thank you for using the Voice Portal, we look forward to hearing you soon. Goodbye.*
	< Set the dialog exit event to `CallerExit`> and < Disconnect the call >

The next dialog flow example demonstrates a voice portal having a phone banking application as one of its sub-services. The voice portal framework is realized with handcrafted VoiceXML pages. The phone banking service is invoked through the processing of a `<subdialog>` element from one of the VoiceXML pages that will hand the dialog processing and control over to the corresponding server instance. When ending the phone banking service through the use of an Exit object, the session on the server ends and, instead of disconnecting the call, the dialog control is handed back to the original VoiceXML page that had triggered the subdialog.

Copyright © 2001-2011 Voxeo Germany GmbH

Object – Caller	Dialog Flow
VoiceXML Page	< Start *Phone Banking* service via `<subdialog>` >
Module	[*Hello and welcome to Phone Banking.*]
Menu	*Main menu! You have the following options* *…* *To leave the Phone Banking service say exit.*
Caller	*Exit*
Exit	*Thank you for using the Phone Banking service.*
	< Set the dialog exit event to `CallerExit` > and < Return to the main dialog >
VoiceXML Page	*Welcome back to the Voice Portal.* *You can choose …*

The third example covers the dialog flow of a customer care voice portal for a mobile operator. Mobile operators divide their subscriber base usually into two distinct groups – prepaid and postpaid. Each group will dial into the same service number as a front door to the self-care portal but will be served with individual call flows according to their subscription status. The dialog flow below shows the concept of service chaining (see also the respective dialog exit type in the Processing section below) where the root service *Front Door* is chained with the *Prepaid / Postpaid* service through an Exit object that is processing the transfer with attached data.

Service	Object – Caller	Dialog Flow
Front Door	Module	[*Welcome to the Mobile Telecom self-care portal.*]
	Connector	< Lookup Prepaid / Postpaid subscriber status and data from the CRM DB based on ANI / MSISDN >
	Exit	< Set dialog exit event to `ServiceChaining` > and < Transfer to the Prepaid / Postpaid service with attached data such as remaining minutes >
< Chain Front Door service with Prepaid service >		
Prepaid	Module	[*There are thirty seven minutes remaining on your prepaid card.*]

Copyright © 2001-2011 Voxeo Germany GmbH

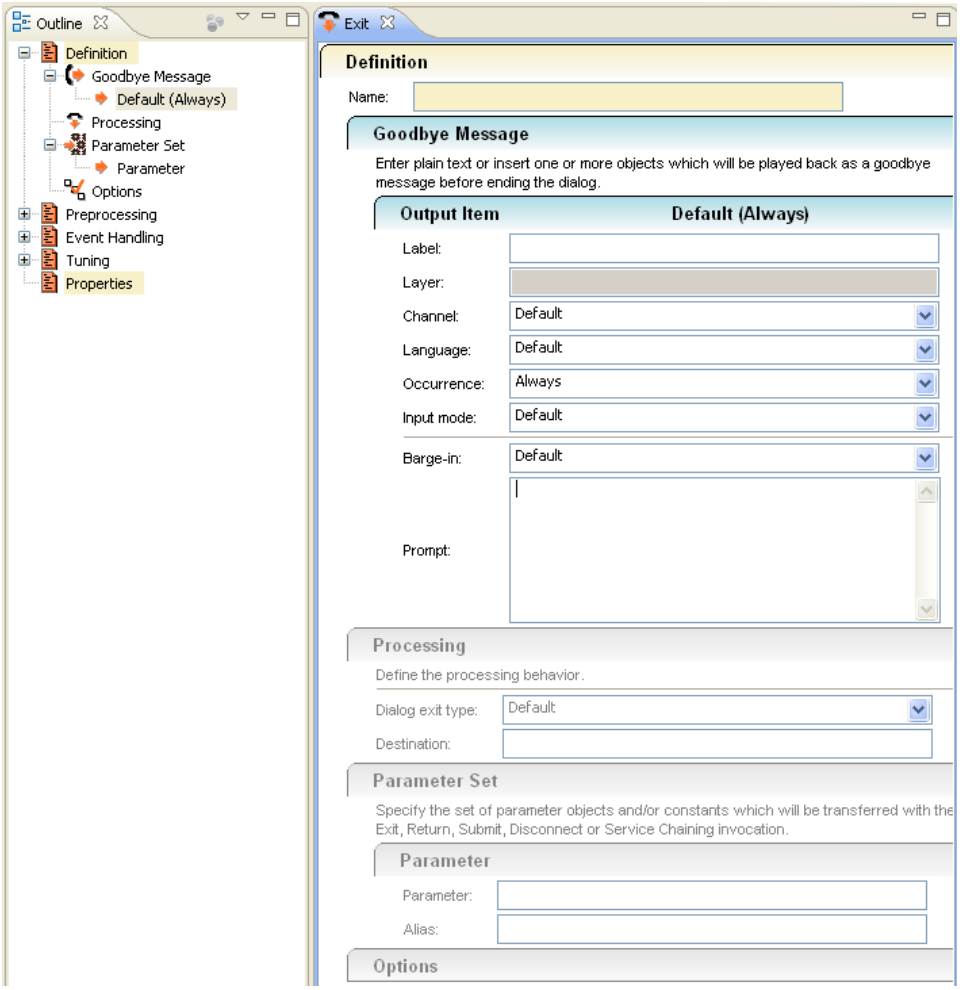
Service	Object – Caller	Dialog Flow
	Menu	[*Main menu! You have the following options.*] … [*To leave the self-care portal say exit at any point in time.*]
	Caller	*Exit*
	Exit	[*Thank you for using the Mobile Telecom self-care service.*]

Object Definition

The **Definition** of the Exit object provides the following sections:

- Goodbye Message
  To define the output informing the caller that the application will end.
- Processing
  To specify the desired type of exit behavior.
- Parameter Set
  To optionally specify parameters to be passed on.
- Options
  To configure processing options.

Copyright © 2001-2011 Voxeo Germany GmbH

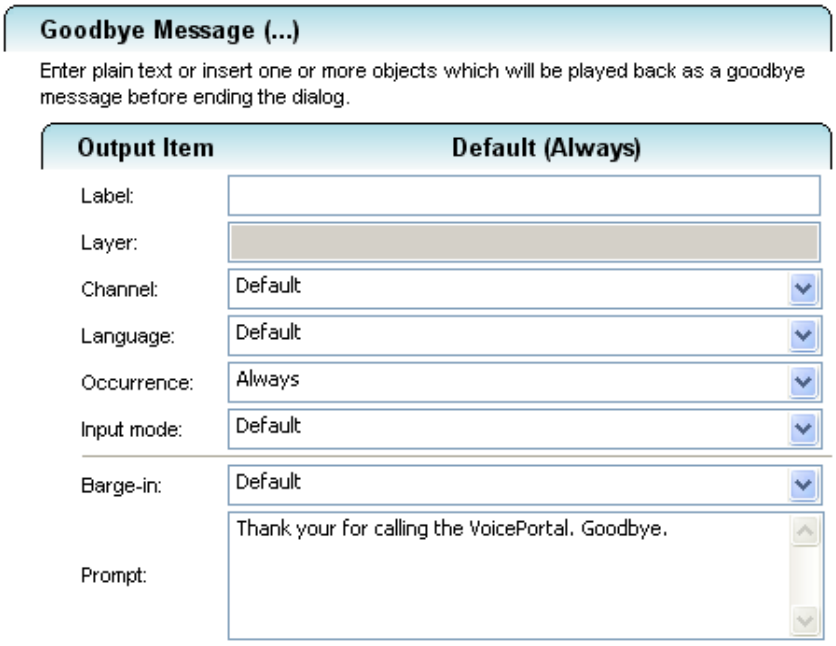


For further details regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Tuning*, and *Properties* in this *Object Reference*.

Goodbye Message

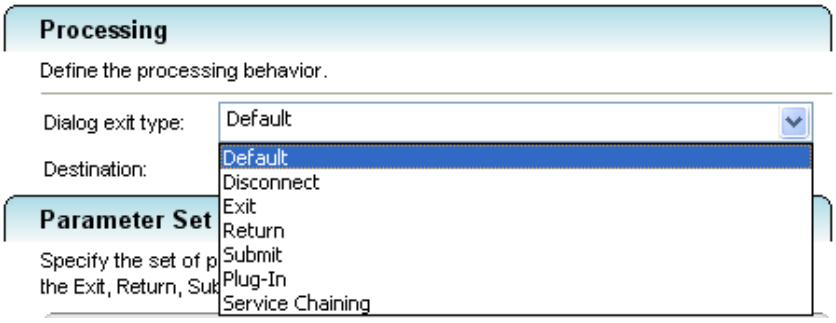
The message specified in the **Goodbye Message** section is played to the caller before the application disconnects the call or returns to the main dialog. It is typically used to inform the caller that the application will end. For further details on output definitions refer to the Output object in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH



Processing

In the **Processing** section, the dialog exit type is defined (see below) as well as the destination for the selected exit type if appropriate. The destination is either a string (for exit types *Service chaining* and *Submit*) or a Plug-In object (for exit type *Plug-In*).



The **Dialog exit type** can be one of the following:

- ***Default***
  Initially, the dialog exit type is set to *Default*, indicating that it is controlled by the corresponding service. For more details on how to set the dialog exit type in the service, refer to Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*.
- ***Disconnect***
  The dialog exit type should explicitly be set to *Disconnect* if the call has to be disconnected when processing the Exit object, independent of the setup of the corresponding service. When this exit type is selected, parameters may optionally be passed to the media platform. See the Parameter Set section for more details.
- ***Exit***
  Hands control over the call back to the media platform, which may either disconnect it or transfer it to some other party (e.g. a Call Center agent). When

Copyright © 2001-2011 Voxeo Germany GmbH

this exit type is selected, parameters may optionally be passed to the media platform. See the Parameter Set section for more details.

- ***Plug-In***
  Processes a Plug-In object as the final step of the dialog. In this case, the plug-in uses its own parameter set as opposed to the one defined in the Exit object. Note that since the dialog session is already terminated, it is not possible to return from the plug-in.
- ***Return***
  Returns call control to a surrounding dialog represented by an external VoiceXML page that initially triggered the processing of the current service by using a `<subdialog>` element. When this exit type is selected, parameters may optionally be passed to the media platform. See the Parameter Set section for more details.
- ***Service Chaining***
  Performs a submit operation to hand over control to another VoiceObjects service. The current service is terminated and the new one started. With this exit type, parameters may optionally be passed to the next service. For convenience the parameters *vsDriver*, *vsLanguage*, *vsInputMode* and *vsRespectUserAgent* are submitted automatically to achieve consistency among the services of the chain. To overwrite them, define them in the parameter set.
  In the **Destination** field, the VSN of the next service must be defined. This implies that the service is hosted on the same VoiceObjects Server. Alternatively, a full URL pointing to a different VoiceObjects environment can be provided. If a full URL is provided, all parameters including the VSN of the next service must be transmitted using the Parameter Set.

  For chained services a master dialog ID is maintained in addition to the dialog ID of the service itself. The master dialog ID is the same across all services of the chain session whereas the dialog ID is specific to one individual service session. It will get logged to Infostore along with other information such as the position of a service in the chain.
- ***Submit***
  Performs a submit operation to hand over control to some other application. When this exit type is selected, parameters may optionally be passed to the media platform. See the Parameter Set section for more details.

When terminating a dialog session through the processing of an Exit object the exit event of the dialog session is always set to the value *ExitObject*. The status of the exit event is available within an Expression object through the function EXITEVENT or within a Script object through the function `vo.exitEvent()`.This information can be retrieved in dialog end processing of a Module object.

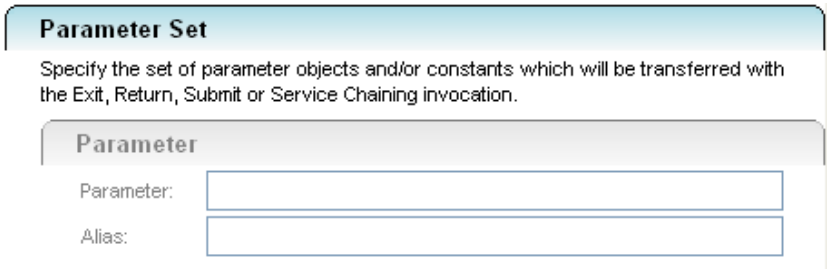
Tip: In the text and Web channel, use Exit objects in all places that you know terminate the dialog. As an example, if you build a text channel application that provides callers with the number of remaining pre-paid minutes, use an Exit object to show this number instead of an Output object. This ensures that the session is immediately freed up instead of having to wait for a session timeout. Additionally, the writing of statistical information for Infostore is optimized.

Parameter Set

In the **Parameter Set** section, **Parameters** for the dialog exit types *Exit*, *Return*, and *Submit*, *Disconnect* and *Service Chaining* are defined which are placed into a name list. The **Alias** field may be used to define the variable name under which the parameter is accessible. Defining an alias is optional. If left empty, the parameter is accessible under its reference ID. This means in particular that for constant values, an alias needs to be defined (as they do not have a reference ID).

Copyright © 2001-2011 Voxeo Germany GmbH

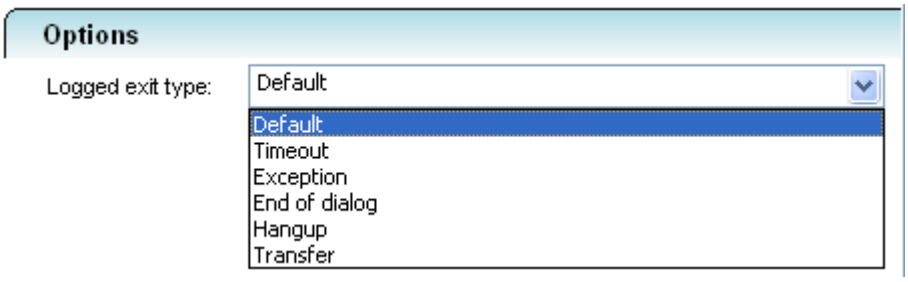
When passing Collection objects, make sure to wrap them with the MASKXML Expression function to ensure that the XML does not break the markup code that is generated.



Note: When using the Exit object to initiate another VoiceObjects service (typically via exit type *ServiceChaining*), the syntax for initial URL parameters described in Service URL Configuration in Chapter 4 – *Service Deployment* in the *Deployment Guide* must be followed. In particular, the prefixes such as *var* (for Variable and Collection objects) and *lay* (for Layer objects) must be used, so that in these cases it is strongly recommended to fill the **Alias** fields. In service chaining, parameters can be transmitted either externally through markup and HTTP requests, or internally within the VoiceObjects Server instance. In the internal case, it is required that the chaining target resides on the same instance as the origin service. The choice between external and internal parameter transfer can be configured on the Service object.

Options

The **Options** section of the **Exit** editor provides one setting that influences which exit type is written to System DB Logging.



Logged exit type defines the exit type that is written to the database. This is useful e.g. when the Exit object is used to initiate a transfer to an agent (e.g. using a Plug-In object) and subsequent reporting expects to see the exit type for the dialog set to *Transfer* to correctly classify it. The default setting of *Default* indicates that no manual adjustment of the exit type is desired, and the one automatically determined by VoiceObjects Server is written to the database.

VoiceObjectsXML Definition

The Exit object is represented by the VoiceObjectsXML element `<exit>`. It has three attributes and six groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The **<**`exit`**>** element uses the embedded **<**`parameterSet`**>** element.

Copyright © 2001-2011 Voxeo Germany GmbH

Exit

Attributes

- **`exitType`**
  Defines the exit type. One of *default*, *disconnect*, *exit*, *plugin*, *return*, *submit*, *serviceChaining*. If not specified, defaults to *default*.
- **`loggedExitType`**
  Defines the exit type written to System DB Logging. One of *default*, *endOfDialog, exception, hangup, timeout* or *transfer*. If not specified, defaults to *default*.
- **`destination`**
  Defines the destination for exit type *submit*. May be a constant string, or a reference to an Expression, Script, or Variable object.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Exit object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Exit object.
- **`<output>`**
  Defines the goodbye message.
- **`<parameterSet>`**
  Defines the parameter set for the Exit object. Only relevant for the exit types *exit*, *return*, *submit, disconnect, serviceChaining*.
- **`<eventHandling>`**
  Defines the event handling for the Exit object.
- **`<tuning>`**
  Defines the tuning settings for the Exit object.

Example

```
<exit exitType="disconnect">
  <output>
    <outputItem><text>Thanks for calling, and
goodbye.</text></outputItem>
  </output>
</exit>
```

ParameterSet

Children

- **`+<item>`**
  Defines the list of entries in the parameter set. The use of the `alias` attribute is optional when the `object` attribute is used, and required when the `value` attribute is used.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<parameterSet>
  <item alias=”customer” object=”#Customer number”/>
  <item alias=”locale” value=”de-DE”/>
</parameterSet>
```

Item

Attributes

- **`object`**
  Defines a reference to an object. If this attribute is defined, the `value` attribute must not be defined.
- **`value`**
  Defines a constant value. If this attribute is defined, the `object` attribute must not be defined.
- **`alias`**
  Defines the parameter alias for the parameter.

Example

```
<item alias=”premium” object=”#Annual premium”/>

<item alias=”locale” value=”en-US”/>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The table below lists all object types that can reference an Exit object:

Icon	Object Name	Use Case Example
	Menu	A common use case for an Exit object is within the Menu object where one Menu item offers the exit of the application.
	Sequence	An Exit object can be used within a Sequence object to automatically end the dialog at a certain point (typically at the end of the sequence).
	If	An Exit object can be used within either the THEN or the ELSE item of an If object.
	Case	An Exit object can be used in any WHEN item and the ELSE item of a Case object.
	Goto	An Exit object can be referenced within a Goto object.
	Hyperlink	The most common use case for an Exit object is within a Hyperlink object where the caller activates the Exit object through a particular voice command.
Any object type providing event handling		The dialog can automatically be ended when a certain event occurs.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of an Exit object should provide the information whether a message is played and how the call will be ended (disconnect vs. return to any other dialog). The following table provides three examples as a general guideline:

Name	Description
 Goodbye & Hang-up	Play a short message to the caller and disconnect the call.
 Disconnect	Disconnect the call without any notification.
 Return to Voice Portal	Return to the main dialog of the voice portal.

Copyright © 2001-2011 Voxeo Germany GmbH

Expression

Overview

The Expression object provides a comprehensive set of expression functions, which can be used to perform arithmetical calculations, string operations, comparison operations, and Boolean operations. In addition there are functions to get access to the dialog context. The expression definition is based on a function and a set of arguments that are specified by constant values or Variable, Collection, Script, Layer, Business Task, or other Expression objects. The result of an Expression object gets dynamically evaluated at call time at the position where it is used within the dialog flow or an object. In a Boolean context the Expression object evaluates to *true* unless the return value is an empty string, 0 (zero), or FALSE (in any combination of upper and lower case letters).

Caution: The list of arguments of an expression function may not have any empty arguments in between, e.g. if a function has four arguments (two fixed plus 2 optional ones) it is not allowed to leave the third argument empty as this will lead to problems while processing the Expression object.

The Object Definition below covers the configuration of the Expression object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Expression object belongs to the object category Logic.

Dialog Flow Scenario

The following example shows an Expression object that checks if the caller's age is below a given value. It is used within an If object that branches the dialog into different flows based on the outcome of the comparison.

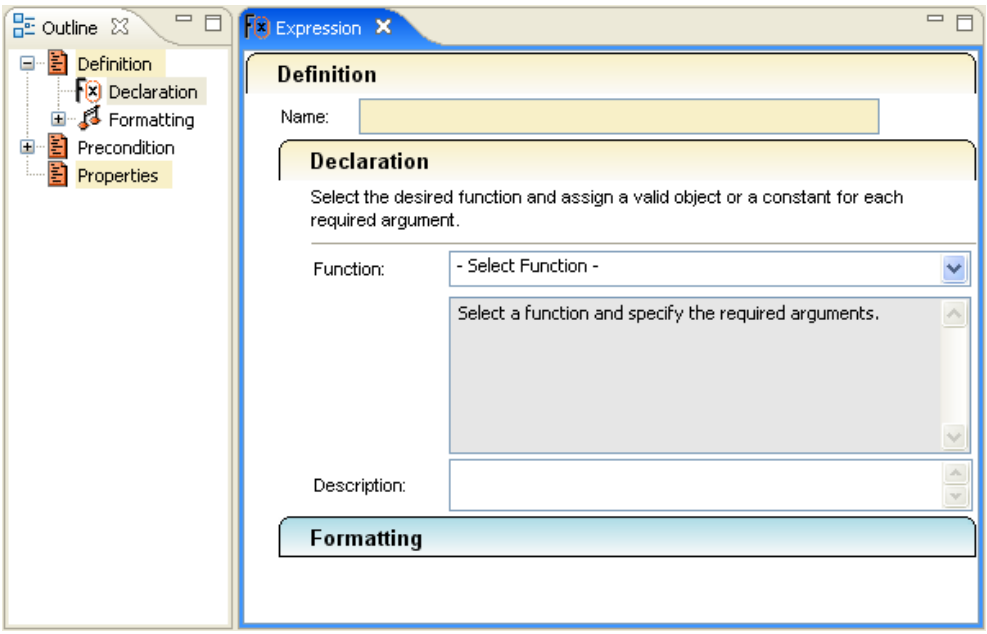
Object - Caller		Dialog Flow
		How old are you?
Caller		*I am sixteen.*
Age<18	THEN:	Inform caller about age restriction
	ELSE:	Proceed with normal dialog flow

Object Definition

The **Definition** of the Expression object provides the following sections:

- Declaration
  To declare the function and assign a valid object or a constant for each required argument.
- Formatting
  To define formatting instructions for playback within an Output.

Copyright © 2001-2011 Voxeo Germany GmbH



For further details regarding additional object configuration refer to *Precondition* and *Properties* in this *Object Reference*.

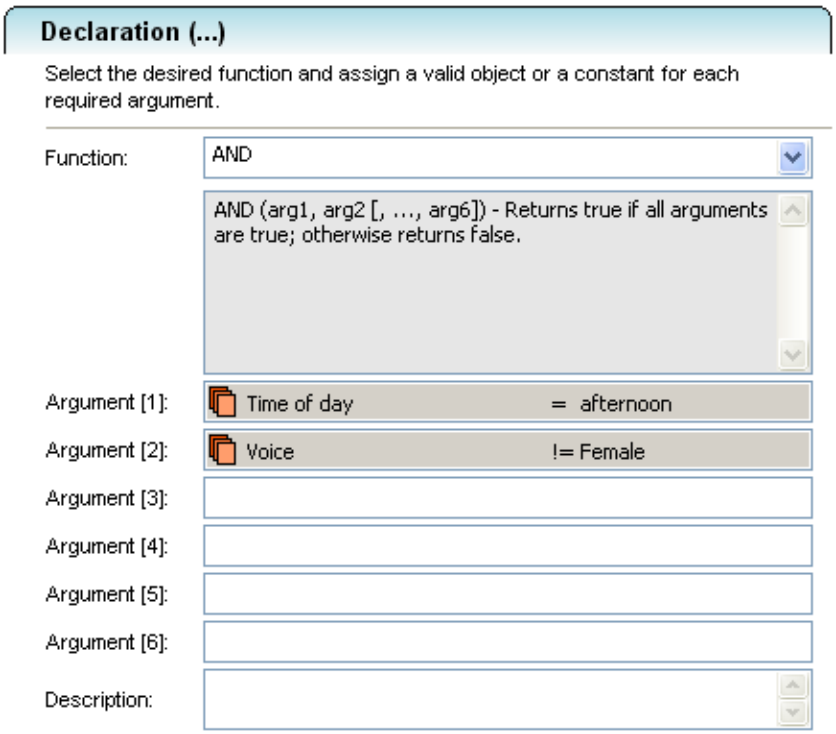
Declaration

Within the **Declaration** section, the desired expression function can be selected from the drop-down list in the **Function** field. In the **Argument** fields the appropriate arguments can be provided as either constant values, or references to Variable, Collection, Script, Layer, Business Task, or other Expression objects. Each expression function provides a brief explanation of the number and types of arguments it requires or accepts in the grey text field below the **Function** field.

A special case of argument is a layer condition that can be used for Boolean functions. For details see Conditions in the function library below.

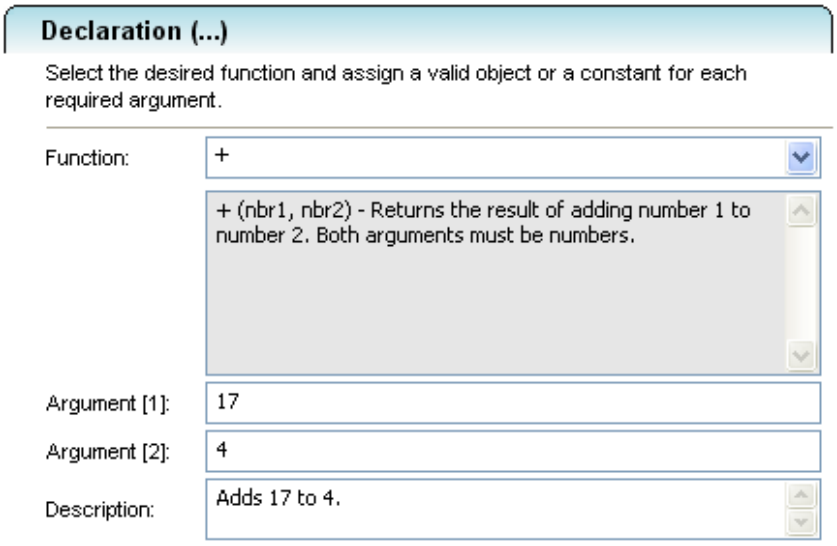
The following example shows the use of the *AND* function that checks the states of two layers, *Time of day* and *Voice*. Only if the layer conditions *Time of day=afternoon* and *Voice!=Female* are true does the function return *true*:

Copyright © 2001-2011 Voxeo Germany GmbH



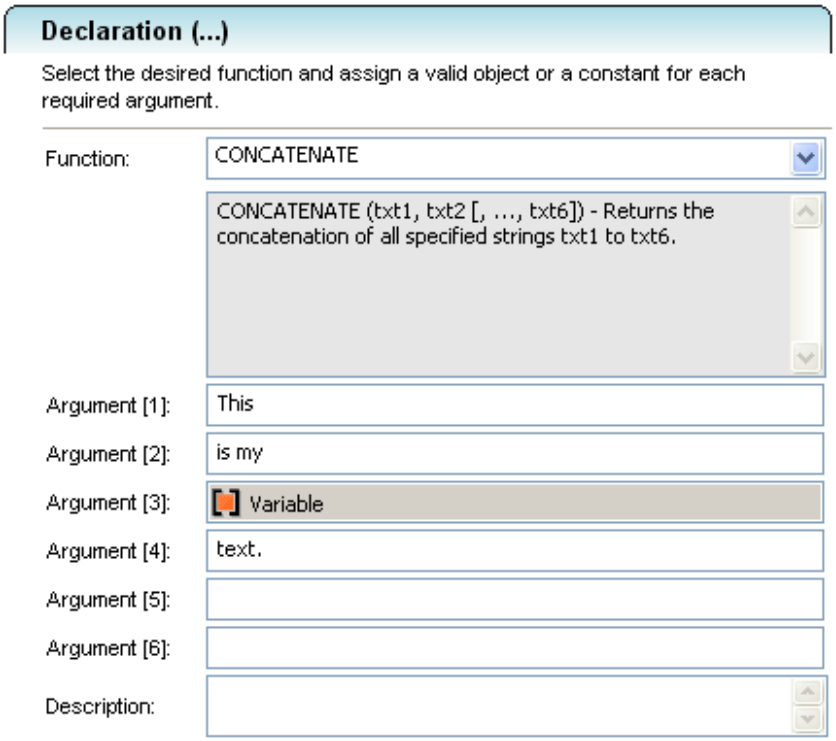
For every expression defined, a brief description can be entered in the **Description** field that explains what the expression does. This description corresponds to the short description for the Expression object (for further information refer to *Properties* in this *Object Reference*).

Each expression function requires or accepts a certain number and certain types of arguments (as stated in the grey text field below). Some functions, e.g. addition and subtraction, require a fixed number of arguments. For each of these required arguments, an entry in the corresponding argument field must be provided.



Copyright © 2001-2011 Voxeo Germany GmbH

Other functions accept optional arguments. For example, the string concatenation accepts between two and six parameters. This means that at least **Arg. [1]** and **Arg. [2]** must be filled, but up to six arguments are allowed as in the example shown below.

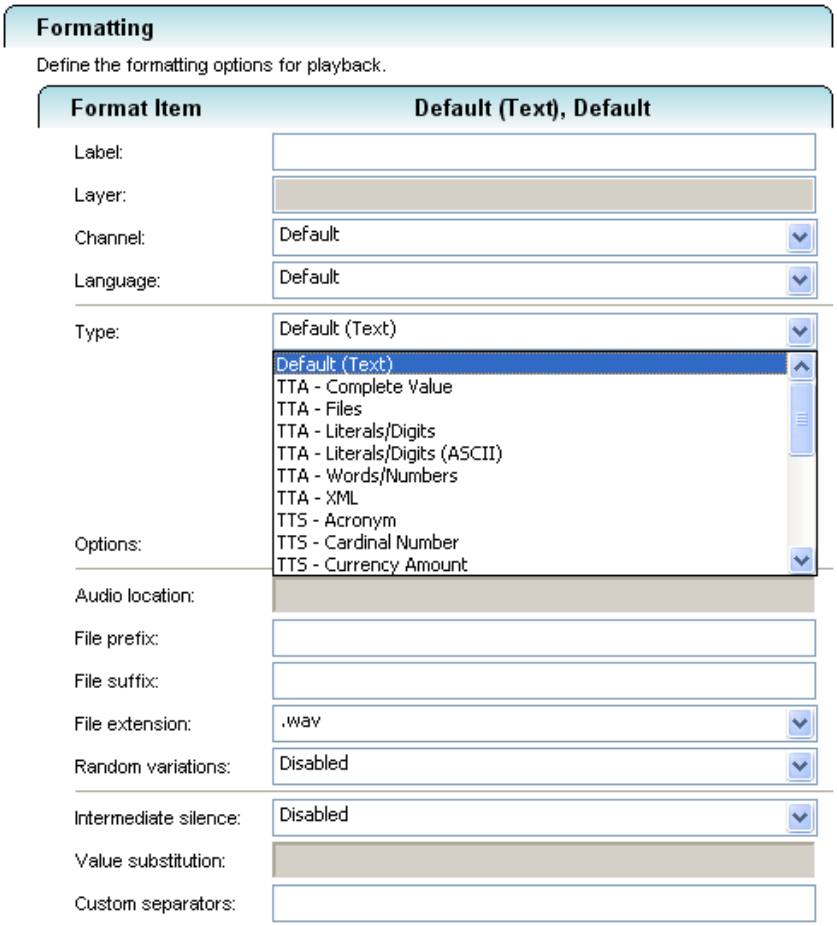


Note: If the evaluation of an Expression object fails, e.g. because a string has been specified in a place where a number is required, resulting errors can be handled within the dialog flow. To do so, specify a dedicated handler for the *Error – Internal* event within the object in which the Expression object is used. For more information refer to *Event Handling* in this *Object Reference*.

Formatting

The **Formatting** section defines how the result of the Expression object is presented to the caller when it is used within an Output object.

Copyright © 2001-2011 Voxeo Germany GmbH



Two different categories of formatting types are offered, which are only relevant for the voice and video channel:

- **Text-To-Audio (TTA)**
  The object content is played back to the caller through the concatenation of prerecorded audio files.
- **Text-To-Speech (TTS)**
  The object content is played back to the caller through text-to-speech synthesis provided by the media platform.

The formatting instructions can be defined as an embedded definition within the Expression object or can be linked to an autonomous Format object.

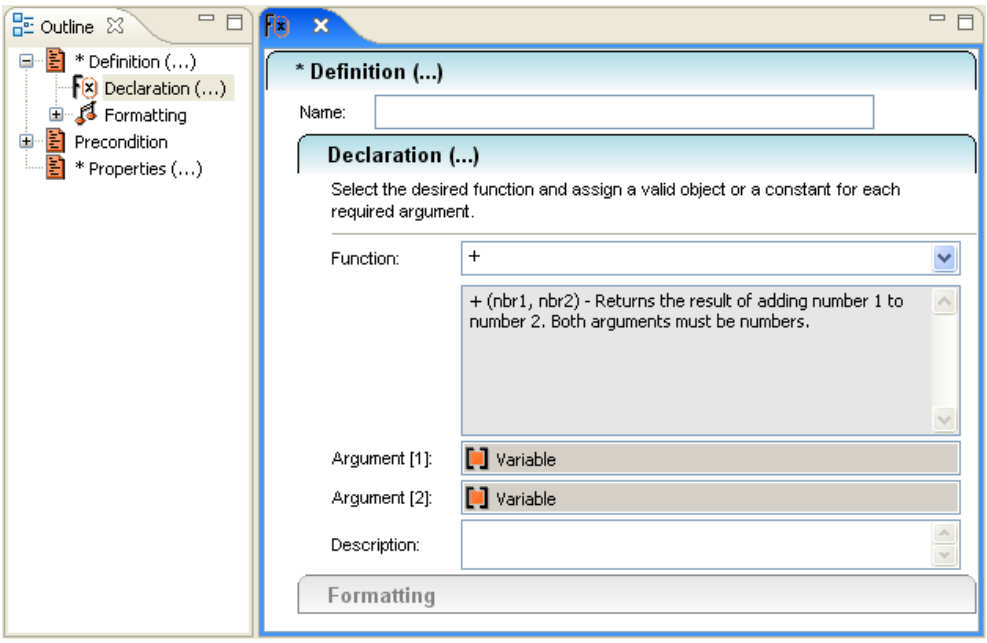
To define formatting types for the text and Web channel, use the Formatting bus as described in Appendix B – *How to Use the Formatting Bus* in the *Administration Guide*.

For further details on formatting capabilities refer to the Format object in this *Object Reference*.

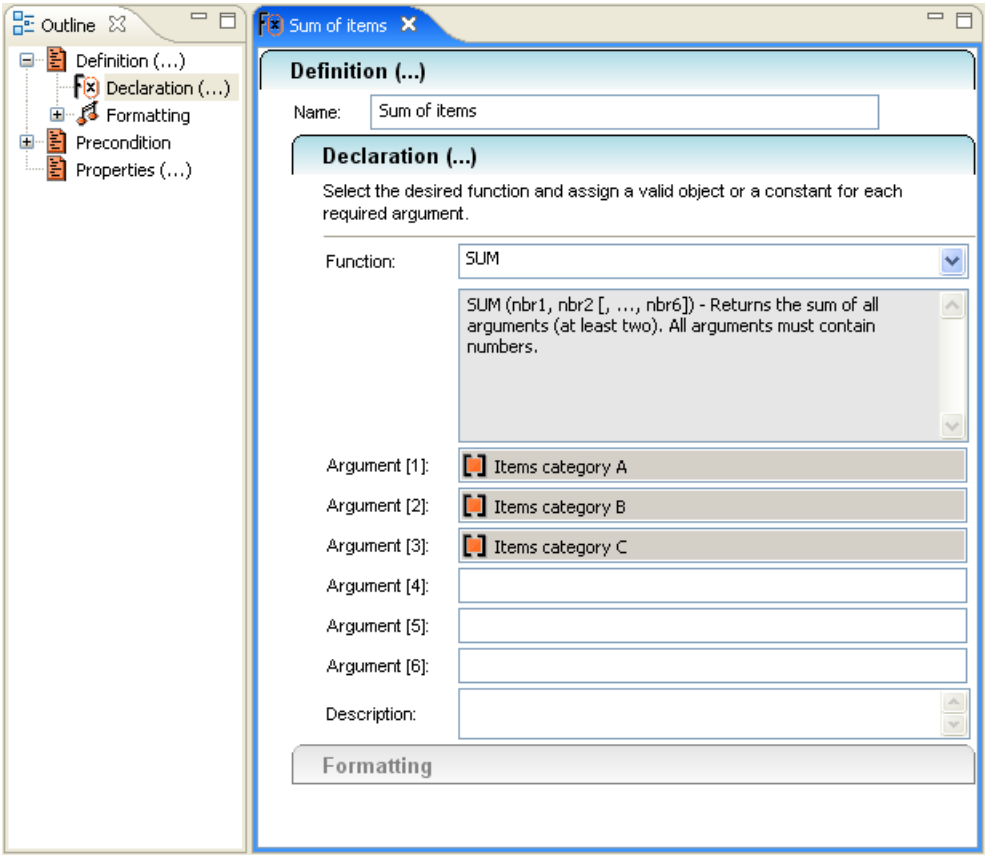
Usage Examples

The first example shows an Expression object that adds two numbers provided by Variable objects.

Copyright © 2001-2011 Voxeo Germany GmbH

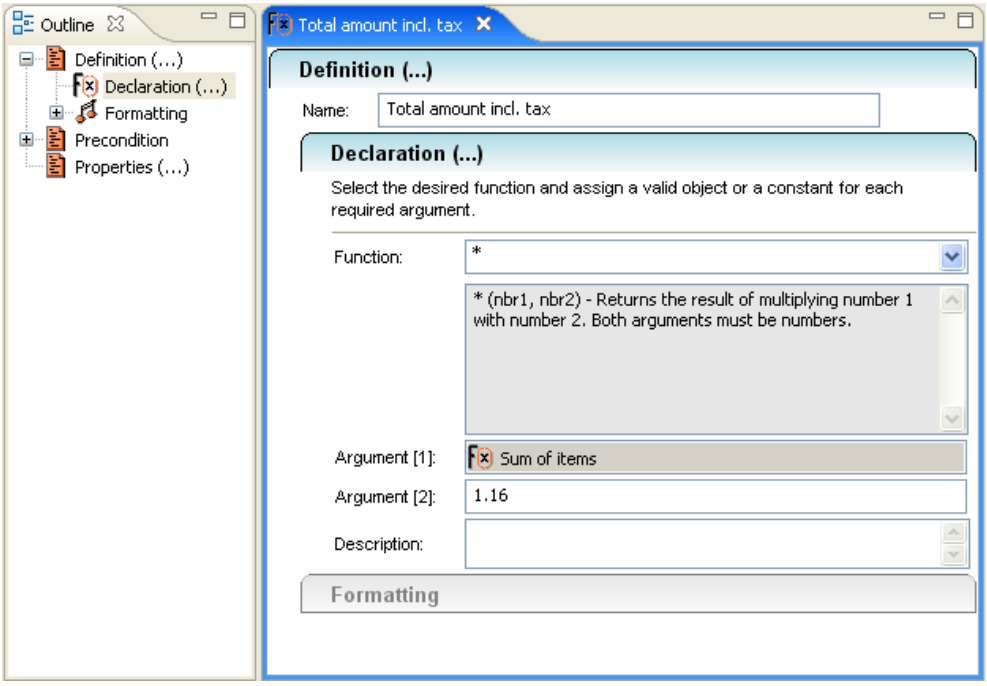


The next example sums up the total amounts of items in three categories A, B, and C. The amounts are stored within the three Variable objects *Items category A*, *Items category B*, and *Items category C*.



Copyright © 2001-2011 Voxeo Germany GmbH

The result of the previous Expression object is used within the next Expression object, which calculates the total sum including tax. The result of the Expression object *Sum of items* is multiplied by the constant value *1.16* to arrive at the gross total amount. This Expression object could then be used e.g. within an Output object that plays the result to the caller.



VoiceObjectsXML Definition

The Expression object is represented by the VoiceObjectsXML element `<expression>`. It has one attribute and three groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<expression>` element uses the embedded `<argument>` element.

Expression

Attributes

- **`function`**
  Defines the expression function. The number of arguments listed as children must match the number of arguments expected by the function specified here.
  Legal functions are listed in the Function Library paragraph, and in the DTD or schema.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Expression object.

Copyright © 2001-2011 Voxeo Germany GmbH

- ***<argument>**
  Lists the arguments for the expression functions; may range from zero to six. The order of the arguments is defined by the order of the children.
- **<format>**
  Defines the output formatting for the expression result.

Example

```
<expression function=”assign”>
  <argument object=”#MyVariable”/>
  <argument><value> 17 </value></argument>
</expression>
```

Argument

Attributes

- **object**
  Object reference as value for the argument. Either this attribute is defined, or the element has a child.
  This attribute may also contain a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.

Children

- **<variable>** or
  <collection> or
  <expression> or
  <script> or
  <layer> or
  <business task>
  Defines an embedded object as argument.
- **<value>**
  Defines the constant value of the argument. If defined, this value takes precedence over the `object` attribute.

Example

```
<argument object=”#Compute total sum”/>
<argument><value> 256 </value></argument>
```

Value

Children

- **CDATA**
  Embedded value for a Variable object, Collection object, or argument.

Example

```
<value> 42 </value>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The Expression object is compatible with any other object type and can be used to dynamically control certain properties or preconditions. The following objects reference Expression objects most frequently:

Icon	Object Name	Use Case Example
	Grammar	An Expression object can be used to dynamically create grammar definitions in a Grammar object.
	Connector	An Expression object can be used within the parameter set in a Connector object when connecting to an external resource.
	Script	An Expression object can be used within the parameter set when processing a Script object.
	Database	An Expression object can be used within the SQL statement definition of a Database object.
	Plug-In	An Expression object can be used within the parameter set when executing custom VoiceXML code in a Plug-In object.
	Log	An Expression object can be used within the log statement definition of a Log object.
	Layer	An Expression object can be used within a Layer object to compute the conditions for the layer states.
	Expression	An Expression object can be used within other Expression objects to build more complex formulas.
	OSDM	An Expression object can be linked within the parameter set of any OSDM object.
Any object with a condition or precondition		An Expression object can be used anywhere as a condition or precondition.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of an Expression object should indicate which type of calculation/processing is done inside the object, typically by mentioning the function name and the arguments. The short description should describe in more detail what the expression does and where it is used, and whether the expression just *returns* a value, or if it *assigns* some value to a Variable or similar object. The table below lists three examples:

Name	Description
 LASTRESULT (confidence)	Returns the confidence value of the last recognition. Used in module *Transfer Money* to evaluate quality of recognition result for decision on whether to transfer money or not.

Copyright © 2001-2011 Voxeo Germany GmbH

Name	Description
ISNULL (Counter)	Returns true if variable *Counter* does not have a value yet.
LAYER (Persona, female)	Sets the layer *Persona* to state *female*.

Function Library

The function library for the Expression object is organized into the following categories:

- Conditions (e.g. Layer condition)
- Comparison functions (e.g. greater than or equal)
- Boolean functions (e.g. AND)
- Information functions (e.g. ISNULL)
- Assign functions (e.g. :=)
- Arithmetic functions (e.g. SQRT)
- Text functions (e.g. CONCATENATE)
- Collection functions (e.g. INDEX)
- Date & Time functions (e.g. NOW)
- Statistics functions (e.g. LASTRESULT)
- Business Task functions (e.g. STARTTASK)
- Dialog Context functions (e.g. SESSION)
- Miscellaneous functions (e.g. CONDITIONAL)

The following tables provide a detailed overview on the available functions and their parameter sets as well as their return values (if any). The functions are sorted by the appropriate function category. An argument list enclosed by normal brackets ( ) indicates a list of mandatory arguments. Square brackets [ ] indicate a list of additional optional arguments.

Conditions		
Name and Arguments		**Description**
Desktop	**XML**	
Layer condition	**layercondition** (arg)	A special function enabling the use of layer conditions as preconditions for objects. Use this function only within the embedded Expression object available as a precondition for most objects. Select or define a complete layer condition *Layer=State* or *Layer!=State* as the function's only argument. For details on how to link a layer refer to Chapter 6 – *Object Editors* in the *Desktop for Eclipse Guide* or *Desktop for Web Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

Conditions		
Name and Arguments		**Description**
Desktop	**XML**	
Channel condition	**channelcondition** (arg)	A special function enabling the use of a channel as a precondition for objects. Use this function only within the embedded Expression object available as a precondition for most objects. Returns *true* if the defined channel is the currently active channel. The argument can be one of *voice*, *video*, *text*, or *web*. More than one channel can be provided, separated by commas.
Input mode condition	**inputmodecondition** (arg)	A special function enabling the use of an input mode as a precondition for objects. Use this function only within the embedded Expression object available as a precondition for most objects. Returns *true* if the defined input mode is the currently active input mode. The argument can be one of *voice*, *dtmf*, or *voicedtmf*. More than one input mode can be provided, separated by commas.
Language condition	**languagecondition** (arg)	A special function enabling the use of a language as a precondition for objects. Use this function only within the embedded Expression object available as a precondition for most objects. Returns *true* if the defined language is the currently active language. The argument must be a valid language code, e.g. *en-US*, *de-DE* etc. More than one language code can be provided, separated by commas.
Occurrence condition	**occurrencecondition** (number, [ReferenceID])	A special function enabling the use of the occurrence count of an object as a precondition. Use this function only within the embedded Expression object available as a precondition for most objects. Returns *true* if the current object or the object with the reference ID provided as the optional second argument occurred less than or equal to the number of times specified in the first argument.

Comparison Functions		
Name and Arguments		**Description**
Desktop	**XML**	
= (arg1, arg2)	**equal** (arg1, arg2)	Returns *true* if argument 1 is equal to argument 2; otherwise returns *false*.
!= (arg1, arg2)	**notequal** (arg1, arg2)	Returns *true* if argument 1 is not equal to argument 2; otherwise returns *false*.

Copyright © 2001-2011 Voxeo Germany GmbH

Comparison Functions		
Name and Arguments		**Description**
Desktop	**XML**	
< (arg1, arg2)	**less** (arg1, arg2)	Returns *true* if argument 1 is less than argument 2; otherwise returns *false*.
<= (arg1, arg2)	**lessequal** (arg1, arg2)	Returns *true* if argument 1 is less than or equal to argument 2; otherwise returns *false*.
> (arg1, arg2)	**greater** (arg1, arg2)	Returns *true* if argument 1 is greater than argument 2; otherwise returns *false*.
>= (arg1, arg2)	**greaterequal** (arg1, arg2)	Returns *true* if argument 1 is greater than or equal to argument 2; otherwise returns *false*.

Note: For all comparison functions, arguments are evaluated as numbers if possible, otherwise as strings. As an example, =(032010, 32010) evaluates to *true*, as both arguments are first converted to the number 32010.

Boolean Functions		
Name and Arguments		**Description**
Desktop	**XML**	
AND (arg1, arg2 [, .., arg6])	**and** (arg1, arg2 [, .., arg6])	Returns *true* if all arguments are true; otherwise returns *false*.
BETWEEN (arg, lowerLimit, upperLimit)	**between** (arg, lowerLimit, upperLimit)	Returns *true* if the specified argument is between the lower and upper limit, or equal to the lower or upper limit; otherwise returns *false*.
CONTAINS (containsTxt, txt, [caseSensitive], [containsTxtIsRegExp])	**contains** (containsTxt, txt, [caseSensitive], [containsTxtIsRegExp])	Returns *true* if *txt* contains *containsTxt* otherwise returns *false*. The optional parameter *caseSensitive* indicates whether the comparison should be case sensitive (*true* or *false*; default is *false*). The optional parameter *containsTxtIsRegExp* indicates whether *containsTxt* has to be interpreted as a string or a regular expression (*true* or *false*; default is *false*).

Copyright © 2001-2011 Voxeo Germany GmbH

Boolean Functions		
Name and Arguments		**Description**
Desktop	**XML**	
ENDSWITH (endTxt, txt, [caseSensitive], [endTxtIsRegExp])	**endswith** (endTxt, txt, [caseSensitive], [endTxtIsRegExp])	Returns *true* if *txt* ends with *endTxt* otherwise returns *false*. The optional parameter *caseSensitive* indicates whether the comparison should be case sensitive (*true* or *false*; default is *false*). The optional parameter *endTxtIsRegExp* indicates whether *endTxt* has to be interpreted as a string or a regular expression (*true* or *false*; default is *false*).
IN (arg, delimitedString, [separator])	**in** (arg, delimitedString, [separator])	Returns *true* if the specified argument is within the list of entries provided within the delimited string; otherwise returns *false*. The default separator is a comma, but a custom separator can optionally be provided as a third argument.
MATCHESREGEXP (txt, regExp, [caseSensitive])	**matchesregexp** (txt, regExp, [caseSensitive])	Returns *true* if *txt* matches the regular expression *regExp*, otherwise returns *false*. The optional parameter *caseSensitive* indicates whether the comparison should be case sensitive (*true* or *false*; default is *false*).
NAND (arg1, arg2 [, .., arg6])	**nand** (arg1, arg2 [, .., arg6])	Returns *true* unless all its arguments are true; otherwise returns *false*. NAND is the logical complement of AND.
NOR (arg1, arg2 [, .., arg6])	**nor** (arg1, arg2 [, .., arg6])	Returns *true* if none of its arguments is true; otherwise returns *false*. NOR is the logical complement of OR.
NOT (arg)	**not** (arg)	Returns the reversed value of its argument.
NOTBETWEEN (arg, lowerLimit, upperLimit)	**notbetween** (arg, lowerLimit, upperLimit)	Returns *true* if the specified argument is not between the lower and upper limit; otherwise returns *false*.
NOTIN (arg, delimitedString, [separator])	**notin** (arg, delimitedString, [separator])	Returns *true* if the specified argument is not within the list of entries provided within the delimited string; otherwise returns *false*. The default separator is a comma, but a custom separator can optionally be provided as a third argument.
OR (arg1, arg2 [, .., arg6])	**or** (arg1, arg2 [, .., arg6])	Returns *true* if any argument is true; otherwise returns *false*.

Copyright © 2001-2011 Voxeo Germany GmbH

Boolean Functions		
Name and Arguments		**Description**
Desktop	**XML**	
STARTSWITH (startTxt, txt, [caseSensitive], [startTxtIsRegExp])	**startswith** (startTxt, txt, [caseSensitive], [startTxtIsRegExp])	Returns *true* if *txt* starts with *startTxt* otherwise returns *false*. The optional parameter *caseSensitive* indicates whether the comparison should be case sensitive (*true* or *false*; default is *false*). The optional parameter *startTxtIsRegExp* indicates whether *startTxt* has to be interpreted as a string or a regular expression (*true* or *false*; default is *false*).
STREQUALS (txt1, txt2, [caseSensitive])	**strequals** (txt1, txt2, [caseSensitive])	Returns *true* if *txt1* and *txt2* are identical otherwise returns *false*. The optional parameter *caseSensitive* indicates whether the comparison should be case sensitive (*true* or *false*; default is *false*).
XOR	**xor** (arg1, arg2 [, .., arg6])	Returns *true* if exactly one argument is true; otherwise returns *false*. XOR (arg1, arg2 [, ..., arg6]) stands for exclusive OR (arg1, arg2 [, ..., arg6]).

Note: Boolean functions do not allow Layer objects as arguments.

Information Functions		
Name and Arguments		**Description**
Desktop	**XML**	
ISAFTER (date1, date2)	**isafter** (date1, date2)	Returns *true* if *date1* occurs chronologically after *date2* otherwise returns *false*. The parameters *date1* and *date2* must be specified in the format yyyyMMdd or yyyyMMddhhmmss.
ISBEFORE (date1, date2)	**isbefore** (date1, date2)	Returns *true* if *date1* occurs chronologically before *date2* otherwise returns *false*. The parameters *date1* and *date2* must be specified in the format yyyyMMdd or yyyyMMddhhmmss.
ISLOGGINGACTIVE (loggingType)	**Isloggingactive** (loggingType)	Returns true if the specified *loggingType* is activated for this call; otherwise returns false. Valid *loggingType* values are *system*, *custom*, *inputState*, *utteranceRecording*, *all*, *none* as well as comma-separated lists thereof.

Copyright © 2001-2011 Voxeo Germany GmbH

Information Functions		
Name and Arguments		**Description**
Desktop	**XML**	
ISLOGICAL (arg)	**islogical** (arg)	Returns *true* if the specified argument is a logical value (*true* or *false*, regardless of capitalization); otherwise returns *false*.
ISNOTNULL (arg)	**isnotnull** (arg)	Returns *true* if the specified argument is defined; otherwise returns *false*.
ISNULL (arg)	**isnull** (arg)	Returns *true* if the specified argument is undefined or empty; otherwise returns *false*.
ISNUMBER (arg)	**isnumber** (arg)	Returns *true* if the specified argument is a number; otherwise returns *false*.
ISTEXT (arg)	**istext** (arg)	Returns *true* if the specified argument is a text; otherwise returns *false*. A number is not interpreted as text, thus ISTEXT(5) returns *false*.
ISVALIDDATE (date)	**isvaliddate** (date)	Returns *true* if date is a valid date otherwise returns *false*. The parameter *date* must be specified in the format yyyyMMdd or yyyyMMddhhmmss.

Assign Functions		
Name and Arguments		**Description**
Desktop	**XML**	
:= (variable, arg)	**assign** (variable, arg)	Assigns the value of the specified argument to the Variable or Collection object.
:= (variable, collection, rowIdx, columnIdx)	**assign** (variable, collection, rowIdx, columnIdx)	Assigns the Collection object entry at row *rowIdx* (a numerical value >= 1) and column *columnIdx* (a numerical or textual value) to the Variable object.
:= (collection, rowIdx, columnIdx, arg)	**assign** (collection, rowIdx, columnIdx, arg)	Assigns the value of the argument to the Collection object entry at row *rowIdx* (a numerical value >= 1) and column *columnIdx* (a numerical or textual value).

Copyright © 2001-2011 Voxeo Germany GmbH

Assign Functions		
Name and Arguments		**Description**
Desktop	**XML**	
:= (collection1, rowIdx1, columnIdx1, collection2, rowIdx2, columnIdx2)	**assign** (collection1, rowIdx1, columnIdx1, collection2, rowIdx2, columnIdx2)	Assigns the Collection object entry in *collection2* at row *rowIdx2* (a numerical value >= 1) and column *columnIdx2* (a numerical or textual value) to the Collection object entry in *collection1* at row *rowIdx1* (a numerical value >= 1) and column *columnIdx1* (a numerical or textual value).
CONDASSIGN (var, cond, truevalue, falsevalue)	**condassign** (var, cond, truevale, falsevalue)	If *cond* is *true*, assigns *truevalue* to the *var* object. Otherwise, assigns *falsevalue*. *cond* can be a Variable, Expression, Script, or Collection object, or a layer condition. *var* can be a Variable, Layer, or Collection object.

Arithmetic Functions		
Name and Arguments		**Description**
Desktop	**XML**	
***** (nbr1, nbr2)	**multiply** (nbr1, nbr2)	Returns the result of multiplying number 1 by number 2. Both arguments must be numbers.
***:=** (variable, nbr)	**multiplyassign** (variable, nbr)	Multiplies the variable value by the specified number and assigns the result to the variable. Both the variable value and the second argument must be numbers.
+ (nbr1, nbr2)	**plus** (nbr1, nbr2)	Returns the result of adding number 1 to number 2. Both arguments must be numbers.
+:= (variable, nbr)	**plusassign** (variable, nbr)	Adds the specified number to the variable value and assigns the result to the variable. Both the variable value and the second argument must be numbers.
- (nbr1, nbr2)	**minus** (nbr1, nbr2)	Returns the result of subtracting number 2 from number 1. Both arguments must be numbers.
-:= (variable, nbr)	**minusassign** (variable, nbr)	Subtracts the specified number from the variable value and assigns the result to the variable. Both the variable value and the second argument must be numbers.

Copyright © 2001-2011 Voxeo Germany GmbH

Arithmetic Functions		
Name and Arguments		**Description**
Desktop	**XML**	
/ (nbr1, nbr2)	**divide** (nbr1, nbr2)	Returns the result of dividing number 1 by number 2. Both arguments must be numbers, and number 2 must not be zero.
/:= (variable, nbr)	**divideassign** (variable, nbr)	Divides the variable value by the specified number and assigns the result to the variable. Both the variable value and the second argument must be numbers. In addition, the second argument must not be zero.
ABS (nbr)	**abs** (nbr)	Returns the absolute value of the specified number. The absolute value of a number is the number without its sign.
AVG (nbr1, nbr2 [, ..., nbr6])	**avg** (nbr1, nbr2 [, ..., nbr6])	Returns the average of all specified arguments.
CEILING (nbr)	**ceiling** (nbr)	Returns a number rounded up, away from zero (0).
EXP (nbr)	**exp** (nbr)	Returns “e” raised to the power of the specified number. The constant “e” equals 2.71828182845904, the base of the natural logarithm.
FLOOR (nbr)	**floor** (nbr)	Returns a number rounded down, toward zero (0).
LN (nbr)	**ln** (nbr)	Returns the natural logarithm of the specified number. Natural logarithms are based on the constant “e” (2.71828182845904). The specified number must be positive.
MAX (nbr1, nbr2 [, ..., nbr6])	**max** (nbr1, nbr2 [, ..., nbr6])	Returns the maximum of all specified arguments.
MIN (nbr1, nbr2 [, ..., nbr6])	**min** (nbr1, nbr2 [, ..., nbr6])	Returns the minimum of all specified arguments.
MOD (nbr, divisor)	**mod** (nbr, divisor)	Returns the remainder after the specified number is divided by the divisor. The result has the same sign as the divisor.
POWER (nbr, power)	**power** (nbr, power)	Returns the value of the specified number to the power specified by the second argument.
RAND( )	**rand** ( )	Returns a Float random number between 0 (inclusive) and 1 (exclusive).
RANDINT(start, end)	**randint** (start, end)	Returns a random Integer value between *start* (inclusive) and *end* (exclusive). Both arguments *start* and *end* must be Integer values, and *end* must be greater than *start*.

Copyright © 2001-2011 Voxeo Germany GmbH

Arithmetic Functions		
Name and Arguments		**Description**
Desktop	**XML**	
ROUND (nbr, nbrDigits)	**round** (nbr, nbrDigits)	Returns the result of rounding the specified number to at most *nbrDigits* decimal digits.
SIGN (nbr)	**sign** (nbr)	Returns the sign of the specified number. Returns 1 if the number is positive, zero (0) if the number is 0, and -1 if the number is negative.
SQRT (nbr)	**sqrt** (nbr)	Returns the square root of the specified number. The number must be positive or zero (0).
SUM (nbr1, nbr2 [, ..., nbr6])	**sum** (nbr1, nbr2 [, ..., nbr6])	Returns the sum of all arguments (at least two). All arguments must contain numbers.
TRUNC (nbr, nbrDigits)	**trunc** (nbr, nbrDigits)	Returns the result of truncating the specified number to at most *nbrDigits* decimal digits.

Text Functions		
Name and Arguments		**Description**
Desktop	**XML**	
APPEND (variable, txt1 [, …, txt5])	**append** (variable, txt1 [, …, txt5])	Appends all provided strings *txt1* to *txt5* to the value of the specified variable.
CONCATENATE (txt1, txt2 [,…, txt6])	**concatenate** (txt1, txt2 [,…, txt6])	Returns the concatenation of all specified strings (at least two).
COUNTOCCURRENCES (countTxt, txt, [caseSensitive], [countTxtIsRegExp]) -	**countoccurrences** (countTxt, txt, [caseSensitive], [countTxtIsRegExp]) -	Returns the number of occurrences of *countTxt* in *txt*. If *countTxt* is not found in *txt* 0 will be returned. The optional parameter *caseSensitive* indicates whether the comparison should be case sensitive (*true* or *false*; default is *false*). The optional parameter *countTxtIsRegExp* indicates whether *countTxt* has to be interpreted as a regular expression (*true* or *false*; default is *false*).
DECODEBASE64 (string, [encoding])	**decodebase64** (string, [encoding])	Returns the decoded version of the string, which is assumed to be base64-encoded. Optionally, the character encoding (e.g. UTF-8) can be provided; otherwise the configured default character encoding is used.

Copyright © 2001-2011 Voxeo Germany GmbH

Text Functions		
Name and Arguments		**Description**
Desktop	**XML**	
ENCODEBASE64 (string, [encoding])	**encodebase64** (string, [encoding])	Returns the base64-encoded version of the string. Optionally, the character encoding (e.g. UTF-8) can be provided; otherwise the configured default character encoding is used.
FIND (findTxt, txt, startPos)	**find** (findTxt, txt, startPos)	Returns the index at which *findTxt* was found within *txt*, starting the search at position *startPos* (where the first character is position 1). If *findTxt* could not be found, 0 (zero) is returned.
INSERT (oldTxt, startPos, newTxt)	**insert** (oldTxt, startPos, newTxt)	Returns a modified version of the string *oldTxt* in which the string *newTxt* has been inserted, starting at position *startPos*.

Copyright © 2001-2011 Voxeo Germany GmbH

<table>
<tr><th colspan="3">Text Functions</th></tr>
<tr><th colspan="2">Name and Arguments</th><th rowspan="2">Description</th></tr>
<tr><th>Desktop</th><th>XML</th></tr>
<tr><td>JSON2XML (json)</td><td>json2xml (json)</td><td>Returns the XML representation of the JSON string json.

As a general rule, all JSON keys are converted to XML elements; attributes are not used. Nameless arrays are wrapped in an element <code><array></code>. If the JSON string results in XML fragments, an outer root element <code><root></code> is automatically wrapped around them.

Examples:

JSON <code>{“first”:1, “second”:2}</code>

XML

<pre><root>
  <first>1</first>
  <second>2</second>
</root></pre>
JSON <code>[“alpha”, “beta”]</code>

XML

<pre><root>
  <array>alpha</array>
  <array>beta</array>
</root></pre>
JSON

<pre>{“person”:
  {“firstName”:”John”,
“lastName”:”Doe”,
  “address”:
    {“street”:”21 2nd Street”,
“zip”:”10021”, city:”New York”,
state:”NY”}
  }}</pre>
XML

<pre><person>
  <firstName>John</firstName>
  <lastName>Doe</lastName>
  <address>
    <street>21 2nd Street</street>
    <zip>10021</zip>
    <city>New York</city>
    <state>NY</state>
  </address>
</person></pre></td></tr>
</table>

Copyright © 2001-2011 Voxeo Germany GmbH

Text Functions		
Name and Arguments		**Description**
Desktop	**XML**	
LEFT (txt, nbrChars, [length], [fillChar])	**left** (txt, nbrChars, [length], [fillChar])	Returns the first *nbrChars* of *txt*. If *nbrChars* is equal to or greater than the length of *txt* the complete string is returned. If *nbrChars* is 0 or negative the empty string is returned. If the optional parameter *length* is set, the return string is guaranteed to be of length *length*. This is achieved by either cutting it or by filling it up to *length* (on the right-hand side) using the *fillChar*. If no *fillChar* is defined, the string is filled up using the blank character.
LEN (txt)	**len** (txt)	Returns the length of the specified string *txt* (number of characters).
LOWER (txt)	**lower** (txt)	Returns a new string in which all uppercase letters of the specified string *txt* are converted to lowercase.
LTRIM (txt, [char])	**ltrim** (txt, [char])	Returns a modified version of the string *txt* in which white space characters (blanks and tabs only) are removed at the left end. Optionally a *char* can be specified which will be removed instead of white space characters.
MASKXML (string)	**maskxml** (string)	Returns a string in which all occurrences of XML-related characters (<, >, ', ", &) are replaced with their corresponding entities.
MID (txt, startPos, nbrChars)	**mid** (txt, startPos, nbrChars)	Returns a substring for the specified string *txt*, starting at position *startPos* for the specified number of characters.
PREFIX (txt, separator, [separatorIsRegExp])	**prefix** (txt, separator, [separatorIsRegExp])	Returns the part of *txt* preceding the first occurrence of *separator*. If *separator* could not be found within *txt* the empty string is returned. The optional parameter *separatorIsRegExp* indicates whether *separator* has to be interpreted as a regular expression (*true* or *false*; default is *false*).
REPLACE (oldTxt, startPos, nbrChars, newTxt)	**replace** (oldTxt, startPos, nbrChars, newTxt)	Returns a modified version of the string *oldTxt* in which the substring starting at position *startPos* containing *nbrChars* characters has been replaced by the string *newTxt*. Example: replace(hello, 3, 2, r) = hero

Copyright © 2001-2011 Voxeo Germany GmbH

Text Functions		
Name and Arguments		**Description**
Desktop	**XML**	
RIGHT (txt, nbrChars, [length], [fillChar])	**right** (txt, nbrChars, [length], [fillChar])	Returns the last *nbrChars* of *txt*. If *nbrChars* is equal to or greater than the length of *txt* the complete string is returned. If *nbrChars* is 0 or negative the empty string is returned. If the optional parameter *length* is set, the return string is guaranteed to be of length *length*. This is achieved by either cutting it or by filling it up to *length* (on the left-hand side) using the *fillChar*. If no *fillChar* is defined, the string is filled up using the blank character.
RTRIM (txt, [char])	**rtrim** (txt, [char])	Returns a modified version of the string *txt* in which white space characters (blanks and tabs only) are removed at the right end. Optionally a [*char*] can be specified which will be removed instead of white space characters.
SPLIT (txt, separator, [separatorIsRegExp])	**split** (txt, separator, [separatorIsRegExp])	Splits *txt* around matches of *separator*, fills the result into a collection and returns this collection. If no matches could be found an empty collection is returned. The optional parameter *separatorIsRegExp* indicates whether *separator* has to be interpreted as a regular expression (*true* or *false*; default is *true*).
STRCOMPARE (txt1, txt2, [caseSensitive])	**strcompare** (txt1, txt2, [caseSensitive])	Compares *txt1* and *txt2* lexicographically and returns 0 if the strings are equal, -1 if *txt1* is lexicographically less than *txt2* and 2 if *txt1* is lexicographically greater than *txt2*. The optional parameter *caseSensitive* indicates whether the comparison should be case sensitive (*true* or *false*; default is *false*).
SUBSTITUTE (string, pattern, replacement, [patternIsRegExp])	**substitute** (string, pattern, replacement, [patternIsRegExp])	Returns a string in which all occurrences of the *pattern* are substituted by the *replacement* string. The original *string* is provided as first argument and will not be modified. The optional parameter *patternIsRegExp* indicates whether *pattern* has to be interpreted as a regular expression (*true* or *false*; default is *true*).
SUFFIX (txt, separator, [separatorIsRegExp])	**suffix** (txt, separator, [separatorIsRegExp])	Returns the part of *txt* following the last occurrence of *separator*. If *separator* could not be found within *txt* the empty string is returned. The optional parameter *separatorIsRegExp* indicates whether *separator* has to be interpreted as a regular expression (*true* or *false*; default is *false*).

Copyright © 2001-2011 Voxeo Germany GmbH

Text Functions		
Name and Arguments		**Description**
Desktop	**XML**	
TRIM (txt, [char])	**trim** (txt, [char])	Returns a modified version of the string *txt* in which white space characters (blanks and tabs only) are removed at the left and right end. Optionally a [*char*] can be specified which will be removed at both ends instead of white space characters.
UNMASKXML (string)	**unmaskxml** (string)	Returns a string in which all occurrences of XML-related entities are replaced with their corresponding characters (<, >, ', ", &).
UPPER (txt)	**upper** (txt)	Returns a new string in which all lowercase letters of the specified string *txt* are converted to uppercase.
URLDECODE(string)	**urldecode**(string)	Returns the URL-decoded version of string.
URLENCODE(string)	**urlencode**(string)	Returns the URL-encoded version of string.
XML2JSON (xml)	**xml2json** (xml)	Returns the JSON representation of the XML string *xml*.

Copyright © 2001-2011 Voxeo Germany GmbH

<table>
<tr><th colspan="3">Text Functions</th></tr>
<tr><th colspan="2">Name and Arguments</th><th rowspan="2">Description</th></tr>
<tr><th>Desktop</th><th>XML</th></tr>
<tr><td>XPATH (XML string, path, [returnAsCollection], [type], [format], [separator])</td><td>xpath (XML string, path, [returnAsCollection], [type], [format], [separator])</td><td>Returns the result of evaluating the XPath expression provided in the second argument on the XML string provided in the first argument. The string can be static, or a reference to a Variable, Collection, Expression, Script, or Layer object. The optional third argument indicates whether the result is returned as a string or as a collection (true or false; default is false).

For more advanced use cases, the optional arguments type, format, and separator may be used. type, which can take values node, nodeset, or string indicates how the result of the XPATH expression should be interpreted. format, which can take values xml, flat, deep, or wrap, indicates how the result should be formatted. And finally separator provides a separation character for listing results (default is blank).

Examples:

XML string:

<root>
<row>
<col name="title">Harry Potter</col>
<col name="price">29.99</col>
</row>
<row>
<col name="title">Lord of the Rings</col>
<col name="price">35.00</col>
</row>
</root>

Path: /root/row/*
Return value: Harry Potter 29.99 Lord of the Rings 35.00

Path: @name
Return value: title price title price

Path: /root/row/col[1]/@name
Return value: title</td></tr>
</table>

Copyright © 2001-2011 Voxeo Germany GmbH

Text Functions		
Name and Arguments		**Description**
Desktop	**XML**	
XSLT(XML string, transformation, [arg3], [arg4], [arg5], [arg6])	**xslt**(XML string, transformation, [arg3], [arg4], [arg5], [arg6])	Transforms the XML provided in the first argument (either as a string or as a URI) by applying the XSL transformation provided in the second argument (also either as a string or as a URI). The optional additional arguments can be used within the XSL transformation. To use the optional parameters, define xsl:params within the transformation, e.g. <xsl:param name="arg3"/> These are filled with the respective argument values and can then be used within the transformation.

Collection Functions		
Name and Arguments		**Description**
Desktop	**XML**	
ADDCOLUMN (destCollection, destColumn, srcCollection, srcColumn)	**addcolumn** (destCollection, destColumn, srcCollection, srcColumn)	Adds a new column with name *destColumn* into the Collection object *destCollection*. The content is copied from the column with name *srcColumn* in Collection object *srcCollection*. Source and destination collection need to have the same number of rows.
ADDROW (destCollection, srcCollection, srcRow)	**addrow** (destCollection, srcCollection, srcRow)	Adds a new row at the bottom of the Collection object *destCollection*. The content is copied from row number *srcRow* in the Collection object *srcCollection*. Source and destination collection need to have the same set of columns.

ADDXMLROW (collection, xmlRow, [position])	**addxmlrow** (collection, xmlRow, [position])	Adds a new row to the collection based on the *xmlRow* definition. Optionally, the position at which to add the row can be provided; by default the new row is added at the bottom.
		The XML format for the row is as in standard collection definitions, i.e.
		
		`<row>`
		`  <col name="..">..</col>`
		`  …`
		`  <col name="..">..</col>`
		`</row>`

Copyright © 2001-2011 Voxeo Germany GmbH

Collection Functions		
Name and Arguments		**Description**
Desktop	**XML**	
CLEAR (collection)	**clear** (collection)	Clears the specified Collection object. Afterwards, the collection is empty.
COLUMNNAMES (resultCollection, collection)	**columnnames** (resultCollection, collection)	Fills the Collection object *resultCollection* with the list of column names present in the second Collection object (one name per row in the column named *column*).
COLUMNS (collection)	**columns** (collection)	Returns the number of columns in the specified Collection object.
DELETECOLUMN (collection, columnIdx)	**deletecolumn** (collection, columnIdx)	Deletes the column with name *columnIdx* from the specified Collection object.
DELETEROW (collection, rowIdx)	**deleterow** (collection, rowIdx)	Deletes the row with index *rowIdx* from the specified Collection object. If the row is not the last one, all rows below the deleted row move up one index position.
FINDCELL (collection, searchColName, searchValue, targetColName, [startRow], [caseSensitiveExact Match])	**findcell** (collection, searchColName, searchValue, targetColName, [startRow], [caseSensitiveExa ctMatch])	Returns the value of the column *targetColName* if the column *searchColName* of the same row has the value *searchValue*. If no such row could be found the empty string is returned. Optionally you can specify a *startRow* to start the search in. If *startRow* is less than 1 or higher than the total number of rows, the search starts at the beginning. The optional parameter *caseSensitiveExactMatch* indicates whether the value comparison should be case sensitive and/or must match exactly (*none*, *both*, *caseSensitive*, *exactMatch*; default is *exact match*).
FINDROW (collection, colName, searchValue, [startRow], [caseSensitive], [exactMatch])	**findrow** (collection, colName, searchValue, [startRow], [caseSensitive], [exactMatch])	Returns the row number of the first row within the specified collection that contains the *searchValue* in the column *colName*. If no row could be found, “0” is returned. Optionally you can specify a *startRow* to start the search in. If *startRow* is below 1 or above the total number of rows, the search starts at the beginning. The optional parameter *caseSensitive* indicates whether the value comparison should be case sensitive (*true* or *false*; default is *false*). The optional parameter *exactMatch* defines if the *searchValue* must match exactly to *colName* (*true* or *false*; default is *true*).

Copyright © 2001-2011 Voxeo Germany GmbH

Collection Functions		
Name and Arguments		**Description**
Desktop	**XML**	
INDEX (collection, rowIdx, columnIdx)	**index** (collection, rowIdx, columnIdx)	Returns the value of the specified collection at row number *rowIdx* and column with name *columnIdx*. If either the row or the column do not exist within the collection, an error is thrown that can be handled by defining an event handler for *Error – Internal* in *Event Handling*.
REPLACECOLUMN (destCollection, destColumnIdx, srcCollection, srcColumnIdx)	**replacecolumn** (destCollection, destColumnIdx, srcCollection, srcColumnIdx)	Replaces the column with name *destColumnIdx* in the Collection object *destCollection*. The content is copied from the column with name *srcColumn* in Collection object *srcCollection*. Source and destination collection need to have the same number of rows.
REPLACEROW (destCollection, destRowIdx, srcCollection, srcRowIdx)	**replacerow** (destCollection, destRowIdx, srcCollection, srcRowIdx)	Replaces row number *destRowIdx* in the Collection object *destCollection*. The content is copied from row number *srcRowIdx* in the Collection object *srcCollection*. Source and destination collection need to have the same set of columns (column names need to be the same).
ROWS (collection)	**rows** (collection)	Returns the number of rows in the specified Collection object.
SORTROWS (collection, columnIdx, direction)	**sortrows** (collection, columnIdx, direction)	Sorts the rows in the specified collection according to the values in the column with name *columnIdx*. If the values in the specified column are all numerical, the sorting uses numerical comparison; otherwise the sorting is alphabetical. Direction can be either *asc* to sort ascending or *desc* to sort descending.
VALUESUBSTITUTION (collection, txt, [languageCode])	**valuesubstitution** (collection, txt, [languageCode])	Searches in the column key within *collection* for the value *txt*. If an exact match is found, the return value is taken from the column whose name matches the current locale of the call (e.g. *en-US* or *de-DE*), if no such column exists for the same row, the value of the column with name "default" is returned. If this column does not exist for this row either, *txt* is returned unchanged. The optional parameter *languageCode* can be used to explicitly define the language code to search for (e.g. *en-US* or *de-DE*; default is the language code of the current call).

Note: For functions to be used for modifying values of a collection refer to the Assign Functions section above.

Copyright © 2001-2011 Voxeo Germany GmbH

Date & Time Functions		
Name and Arguments		**Description**
Desktop	**XML**	
ADDDATE (date, increment, [unit])	**adddate** (date, increment, [unit])	Adds *increment* to *date* and returns the new date in the format yyyyMMddhhmmss. The parameter *date* must be specified in the format yyyyMMdd or yyyyMMddhhmmss. If increment is less than 0, *date* is decreased accordingly. The optional parameter *unit* can be used to define the unit of *increment* as one of *years*, *months*, *days*, *hours*, *minutes*, *seconds* (default is *days*).
CONVERTDATE (txt, outputFormat, [inputFormat], [languageCode])	**convertdate** (txt, outputFormat, [inputFormat], [languageCode])	If *txt* is a date in the format yyyyMMdd or yyyyMMddHHmmss this returns the date in the specified *outputFormat* using the Java SimpleDateFormat convention. If *txt* is a date following a custom format the optional parameter *inputFormat* must be used to specify the format syntax in order to enable the correct conversion (e.g. if *txt* contains "23102008" the *inputFormat* should be set to "ddMMyyyy"). The optional parameter *languageCode* can be used to define the language code based on which the regional setting is determined (e.g. *en-US* or *de-DE*; default is the language code of the current call). This information will be used to localize the returned date format if necessary, e.g. if the call is processed in *en-US* a date may contain the abbreviation *AD*, when switching the language code to *de-DE* this would be changed to *n.Chr.*
LASTDAYINMONTH (date)	**lastdayinmonth** (date)	Returns the last day in the month represented by *date* in the format yyyyMMddhhmmss. The parameter *date* must be specified in the format yyyyMMdd or yyyyMMddhhmmss.
NEXTWEEKDAY (date, weekday, [languageCode])	**nextweekday** (date, weekday, [languageCode])	Returns the date of the next weekday, which is represented by an integer from 1 (first day in week) to 7 (last day in week), in the format yyyyMMddhhmmss starting from *date*. The parameter *date* must be specified in the format yyyyMMdd or yyyyMMddhhmmss. Depending on the regional setting this may differ as the week in Europe typically starts with Monday, whereas the US week starts with Sunday. The optional parameter *languageCode* can be used to define the language code based on which the regional setting is determined (e.g. *n-US* or *de-DE*; default is the language code of the current call).

Copyright © 2001-2011 Voxeo Germany GmbH

Date & Time Functions		
Name and Arguments		**Description**
Desktop	**XML**	

NOW ([format], [offsetToUTC])	**now** ([format], [offsetToUTC])	Returns the current timestamp in the specified format using the Java SimpleDateFormat convention. Sample formats and return values are:
		`yyyy.MM.dd G 'at' HH:mm:ss z`
		`2001.07.04 AD at 12:08:56 PDT`
		`EEE, MMM d,''yy`
		`Wed, Jul 4,'01`
		`h:mm a`
		`12:08 PM`
		`hh 'o''clock' a, zzzz`
		`12 o'clock PM, Pacific Daylight Time`
		`K:mm a, z`
		`0:08 PM, PDT`
		`yyyyy.MMMMM.dd GGG hh:mm aaa`
		`02001.July.04 AD 12:08 PM`
		`EEE, d MMM yyyy HH:mm:ss Z`
		`Wed, 4 Jul 2001 12:08:56 -0700`
		`YyMMddHHmmssZ`
		`010704120856-0700`
		If no format is specified, the default format **`yyyyMMddHHmmss`** is used.
		The optional second parameter offsetToUTC can be used to switch to a different time zone when interpreting the format parameter. The offset must be provided in the format [-]HHMM.
		A detailed definition can be found at: *http://java.sun.com/javase/6/docs/api/java/text/SimpleDateFormat.html*

TIMEBETWEEN (date1, date2, [unit])	**timebetween** (date1, date2, [unit])	Returns the number of days between *date1* and *date2*. The parameters *date1* and *date2* must be specified in the format yyyyMMdd or yyyyMMddhhmmss. The optional parameter *unit* can be used to define the unit as one of *years*, *months*, *days*, *hours*, *minutes*, *seconds* (default is *days*). The calculation ignores everything below *unit* (e.g. if *unit* is set to hours, minutes and seconds of *date1* and d*ate2* will be ignored).
TIMER (option)	**timer** (option)	Provides a stopwatch functionality. When called with argument *start* it resets the timer. When called with argument *interval* it returns the time difference between now and the last *start* or *interval* request (in milliseconds).

Copyright © 2001-2011 Voxeo Germany GmbH

Date & Time Functions		
Name and Arguments		**Description**
Desktop	**XML**	
TIMEZONEINFO ()	**timezoneinfo** ()	Returns a collection with four columns. The first column *Offset* shows the offset in hours to add to UTC to get local time (e.g. "-0500" or "0230"). The second column *StandardName* contains the standard name of the local timezone (e.g. *PST* or *CET*). The third column *DaylightName* shows the daylight name of the local time zone (e.g. *PDST* or *CEDT*). The fourth column *DSTInEffect* indicates whether the local time is Daylight Saving Time or not (*true*, *false* or *unknown*).
WEEKDAY (date, [languageCode])	**weekday** (date, [languageCode])	Returns the weekday of *date* represented by an integer from 1 (first day in week) to 7 (last day in week). The parameter *date* must be specified in the format yyyyMMdd or yyyyMMddhhmmss. Depending on the regional setting this may differ as the week in Europe typically starts with Monday, whereas the US week starts with Sunday. The optional parameter *languageCode* can be used to define the language based on which the regional setting is determined (e.g. *en-US* or *de-DE*; default is the language code of the current call).

Note: A number of date- and time-related functions (ISAFTER, ISBEFORE, ISVALIDDATE) can also be found in the **Information Functions** group.

Statistics Functions		
Name and Arguments		**Description**
Desktop	**XML**	
LASTTRANSITION (value)	**lasttransition** (value)	Returns the mode (for *value* = *mode*) or type (for *value* = *type*) of the most recent dialog transition. Possible modes are *Standard Navigation*, *Auto Advance*, *Recognition*, *Hyperlink*, *Hangup*, and *Error*. The type holds more fine-grained information on how the caller left the last input state.

Copyright © 2001-2011 Voxeo Germany GmbH

Statistics Functions		
Name and Arguments		**Description**
Desktop	**XML**	
LASTRESULT (type, [index], [phase])	**lastresult** (type, [index], [phase])	Returns information on the most recent recognition result, as well as information on results from recordings and transfers. The first argument can be *confidence*, *interpretation*, *length*, recordingpath, or recordingurl (for recognition results)*, inputmode* (for recognition results or transfers), *utterance* (for recognition results, transfers, and recordings), *markname* or *marktime* (for prompt markers and barge-in point detection), *size* or *maxtime* (for recordings), *duration* (for recordings and transfers), *transferstatus* (for transfers), and *validation* (for recognition results with server-side input validation, using the regex: … :regex syntax for TTG grammars in the text and web channels). For recognition results, if the recognizer returns more than one result, the optional second argument can be used to specify the desired result. *Index* must then be a number between 1 and *Max N-Best* (a tuning property). If *index* is greater than the maximum number of results, the function returns an empty string. The type *length* returns the number of results. The third argument can be used to distinguish between the *collection* (the default) and *confirmation* phase in OSDM objects. If it is set, the second argument must also be defined. *Markname* returns the name of the last prompt marker processed (a `<mark>` statement in an Output item). *Marktime* returns the time elapsed since the last marker was processed. For more information on the recording and transfer related types, see the Recording and Transfer object documentation. *Recordingpath* returns the path and *recordingurl* returns the URL of the last recorded utterance, in case utterance recording was enabled. *Validation* returns a comma-separated list of all slot names for which input validation (against the regular expression for that slot) failed. When called with a slot name as the second argument, the return value is *true* if validation was successful, *false* if it failed.
EVENTLIST (scope, [phase])	**eventlist** (scope)	Returns a comma-separated list of events, which have occurred either in the most recent input state (*scope* = *local*) or in the entire dialog (*scope* = *global*). The second argument allows distinguishing between the *collection* and *confirmation* phase of OSDM objects. The default value is *global*, covering both phases.

Copyright © 2001-2011 Voxeo Germany GmbH

Statistics Functions		
Name and Arguments		**Description**
Desktop	**XML**	
EVENTCOUNTER (event, [scope], [phase])	**eventcounter** (event, [scope])	Returns the number of times the specified event has occurred during the most recent input state (*scope* = *local*) or during the entire dialog (*scope* = *global*). If the scope is not defined, it defaults to *global*. The third argument allows distinguishing between the *collection* and *confirmation* phase of OSDM objects. The default value is *global*, covering both phases. Example: *eventCounter(nomatch)* will return the number of No Match events that have been triggered in the current dialog. For more information on the different events, refer to Event Handling in this *Object Reference*.

Business Task Functions		
Name and Arguments		**Description**
Desktop	**XML**	
STARTTASK (task)	**starttask** (task)	Starts the business task that is specified in the first argument.
FINISHTASK (task, status)	**finishtask** (task, status, clearNavigationHistory)	Finishes the business task that is specified in the first argument with the status specified in the second argument. This can be one of *backEndError*, *businessLogic*, *callerAbort*, *recognitionFailure*, *technicalError*, or *complete*. Optionally, the first argument can be *all* to finish all tasks or *active* to finish the currently active task. All values are case-sensitive. The optional parameter *clearNavigationHistory* defines if the navigation history should be reset or not (true or false; default is false).
TASKREFID ()	**taskrefid** ()	Returns the reference ID of the currently active business task. If no task is active an empty string is returned.
TASKSTATUS (task)	**taskstatus** (task)	Returns the most recent status of the business task specified in the first argument. If the task has not been finished yet, it returns *active* or *inactive*. If the task has never been started, it returns *notStarted*.

Copyright © 2001-2011 Voxeo Germany GmbH

Business Task Functions		
Name and Arguments		**Description**
Desktop	**XML**	
LASTTASK([mode])	**lasttask**([mode])	Returns information about the last business task processed within the current dialog session. Valid modes are: • **completion** Completion result of the business task; either *complete* or *incomplete* • **duration** Duration of the business task in milliseconds • **occurrence** Occurrence of the same business task within this dialog session • **refID** Reference ID of the business task • **status** Status in which the business task was finished • **steps** Number of input steps processed while the business task was active. This information is only available if System DB logging is active for the service. If no business task has been finished so far, the empty string is returned for all modes.

Dialog Context Functions		
Name and Arguments		**Description**
Desktop	**XML**	
ACTIVATEEVENT (event, [return])	**activateevent** (event, [return])	Activates a custom event. The obligatory first argument is the name of the event to be activated. A corresponding handler must be defined in a Module object. Note that occurrence settings are ignored. The optional second argument defines if processing will continue after the Expression object (*true*), or not (*false*). If not defined, defaults to *true*. For more information on the event types, refer to *Event Handling* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions		
Name and Arguments		**Description**
Desktop	**XML**	
APPLYCONFIGURATION (configurationXML)	**applyconfiguration** (configurationXML)	Applies the assignments defined in *configurationXML*. The argument must be a well-formed XML string, not a reference to a file. The XML format used is the same as for application defaults. For further details refer to Application Defaults in Chapter 4 – *Service Deployment* in the *Deployment Guide*.
BARGEIN ([arg])	**bargein** ([arg])	When used without an argument, indicates whether barge-in is activated by default (*true*) or not (*false*). When used with an argument (either *true* or *false*), it sets the default barge-in behavior.
BREAK()	**break**()	Only applicable within a Loop object. Skips the rest of the objects defined in the loop sequence and continues behind the Loop object.
CALLID ( )	**callid** ( )	Returns the unique ID of the call as a string of 40 characters.
CHANNEL ( )	**channel** ( )	Returns the code of the channel (voice, video, text, Web) currently used for this service.
CLEARNAVIGATIONHISTORY ()	**clearnavigationhistory** ()	Clears the standard navigation history.
CONTINUE()	**continue**()	Only applicable within a Loop object. Skips the rest of the objects defined in the loop sequence and continues with the next loop condition evaluation.
DATABASETYPE ( )	**databasetype** ( )	Returns the type of the database used as VoiceObjects Custom DB logging repository, e.g. Oracle or *SQLServer*.
DATABASEURL ( )	**databaseurl** ( )	Returns the URL that is required to connect to the VoiceObjects Custom DB logging repository.
DIALOGID ( )	**dialogid** ( )	Returns the unique ID of the dialog as a string of 44 characters.
DIALOGSTEPID ()	**dialogstepid** ()	Returns the index of the current input state in the dialog, starting with 0. Before the first caller interaction, it returns -1. When Infostore logging on input state level is enabled, the return value matches the entry of column VOLDDSSEQ.DS_STEP for the most recently processed input state.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions		
Name and Arguments		**Description**
Desktop	**XML**	
DRIVER ( )	**driver** ( )	Returns a string that represents the media platform driver information of the corresponding service (e.g. Nuance_VWS_2_0_VXML_2_0 or IBM_WVS_2_0_VXML_1_0).
DRIVERID ()	**driverid** ()	Returns the ID of the media platform driver of the corresponding service.
DRIVERASRENGINE ( )	**driverasrengine** ( )	Returns a string that represents the name of the ASR engine (e.g. *Nuance* or *IBM*).
DRIVERPRODUCT NAME ( )	**driverproductname** ( )	Returns a string that represents the media platform product name of the corresponding service (e.g. *VWS* or *OCSW*).
DRIVERPRODUCT VERSION ()	**driverproductversion** ()	Returns a string that represents the media platform product version of the corresponding service (e.g. 1.2 or 2.0).
DRIVERVENDOR NAME ()	**drivervendorname** ()	Returns a string that represents the media platform vendor name of the corresponding service (e.g. *Nuance* or *IBM*).
DRIVERVXML VERSION ( )	**drivervxmlversion** ( )	Returns a string that represents the VoiceXML version of the corresponding service (e.g. 1.0 or 2.0).
ERRORCODE ( )	**errorcode** ( )	Returns the error code of the most recently processed Connector item, or zero.
ERRORMESSAGE ( )	**errormessage** ( )	Returns the error message of the most recently processed Connector item, or the empty string.
EXITEVENT ( )	**exitevent** ( )	Returns a string providing the corresponding exit event ending the dialog. Valid return values are: *None, Timeout, Exception, EndOfDialog, CallerHangup, CallerExit,* and *Transfer*. For information on how to exploit this function in dialog end processing, refer to the Module object in this *Object Reference*.
GRAMMARCONTROL ([arg])	**grammarcontrol** ([arg])	When used without an argument, indicates whether grammar-driven application control is activated by default (*true*) or not (*false*). When used with an argument (either *true* or *false*), it sets the default grammar control behavior.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions		
Name and Arguments		**Description**
Desktop	**XML**	
GRAMMARTYPE ([type])	**grammartype** ([type])	When used without an argument, returns the code of the currently active grammar type (one of *gsl*, *jsgf*, *sgrs_xml*, *precompiled*, *regex*, *abnf*, *none*). When used with an argument, it sets the currently active grammar type to this value. The argument must be one of the grammar type codes listed above. Note that the underlying media platform must be configured appropriately to support the grammar type that is being set.
INIT (arg)	**init** (arg)	Resets a variable or collection to its initial value, which was determined when it was first used. Note: If the initial value is not specified, the variable or collection is not reset to an empty string, but to the first value it received during the current call.
INPUTMODE ([mode])	**inputmode** ([mode])	When used without an argument, returns the code of the currently active input mode (*voice*, *dtmf*, or *voicedtmf*). When used with an argument, it sets the currently active input mode to this value.
ITERATE (counter, from, to, expression)	**îterate** (counter, from, to, expression)	Iterates *counter* from *from* to *to* and processes *expression* in each iteration. *Counter* must be a reference to a Variable object, which can then be read out within *expression* for further actions. *From* and *to* must be integer numbers. *Counter* will run through the values *from*, *from+1*, *to*.
LANGUAGE ([languageCode])	**language** ([languageCode])	When used without an argument, returns the code of the currently active language (e.g. *en-US* or *de-DE*). When used with an argument, it sets the currently active language to this value. Note that the underlying media platform must be configured appropriately to support the language that is being set. Refer to Appendix A - *Language Codes* in this *Object Reference* for a list of all currently supported language codes.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions		
Name and Arguments		**Description**
Desktop	**XML**	
LAYER (layer, [state])	**layer** (layer, [state])	When used without a second argument, returns the comma-separated list of active states for the specified layer. When used with a second argument, it sets the active state of the layer to this value. The value must be one of the defined states of the layer. Only manual layers can be set in this way. There are three special values for the second argument: ACTIVE returns the comma-separated list of currently active states for the layer; INACTIVE returns the comma-separated list of currently inactive states; ALL returns the comma-separated list of all states.
MASTERDIALOGID ( )	**masterdialogid** ( )	Returns the unique master dialog ID of the session as a string of 44 characters. The master dialog ID is used when chaining services. If the current service is not part of a chain, this ID is the same as the dialog ID.
MD5SUM (txt)	**md5sum** (txt)	Returns the 128 bit MD5 hash of *txt* as a sequence of 32 hexadecimal digits. Hashing is performed according to RFC 1321, e.g. supplying the string *VoiceObjects* will result in the following hash value: 31414f01914a0366ef9f247bb2eaa9f5.
MODULEHISTORY ([level])	**modulehistory** ([level])	Returns a comma-separated list of reference IDs of Module objects visited so far in the current dialog. When used without an argument the entire list of all visited Module objects will be returned, with the most current Module object at the top of the list. When used with an argument the value of the argument defines how far to go back in the processing history specifying where to start the list (e.g. *level* = 1 means that the list starts with the previously processed Module object instead of the current one). If the specified level exceeds the size of the processing history an empty string is returned. Note: Only Module objects with the option *Enable history tracking* set to *true* are considered.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions		
Name and Arguments		**Description**
Desktop	**XML**	
MODULEREFID ([level])	**moduleref id** ([level])	Returns the reference ID of a Module object processed in the current dialog. When used without an argument the reference ID of the currently processed Module object is returned. When used with an argument ([level]) the reference ID of a parent object of the currently processed Module object is returned. The value of the argument specifies how far to go up in the tree of parent modules (e.g. *level* = 1 means that the reference ID of the first parent of the currently processed Module object is returned). If the specified level exceeds the depth of the module tree an empty string is returned.
MODULESET ([level])	**moduleset** ([level])	Returns a comma-separated list of reference IDs of Module objects processed so far in the current dialog. The list is sorted alphabetically, and each module only occurs once. When used without an argument the entire list will be returned. When used with an argument the value of the argument defines how far to go back in the processing history specifying where to start the list (e.g. *level* = 1 means that the list starts with the previous Module object instead of the current one). If the specified level exceeds the size of the processing history an empty string is returned. Note: Only Module objects with the option *Enable history tracking* set to *true* are considered.
NOTIFICATION (message)	**notification** (message)	Sends a notification with a predefined ID (1.3.6.1.4.1.24140.1.2.5.8.1) and the specified message.
OUTPUTMODE ([mode])	**outputmode** ([mode])	When used without an argument, returns the code of the currently active output mode (*audio:tts* or *tts:audio*). When used with an argument, it sets the currently active output mode to this value.
PROCESSFORMATTING (object)	**processformatting** (object)	Processes the formatting of *object* and returns a collection of TTA-XML (also in case one of the standard format types has been selected). The generated markup code can be retrieved by extracting the value of the column markup. *Object* can be a reference to a Variable, Collection, Expression, Script, or Layer object.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions		
Name and Arguments		**Description**
Desktop	**XML**	
PROJECTINFO (mode)	**projectinfo** (mode)	Provides information on the project that contains the start object of the current service. For mode *project*, it returns the name of the project. For mode *version*, it returns the name of the project version.
PRONUNCIATION (var)	**pronunciation** (var)	Returns the pronunciation pattern associated with the variable provided as first argument.
REFRESH (arg)	**refresh** (arg)	Re-evaluates the initial value of a variable or collection in the current dialog context. For a static initial value, this is the same as INIT(). Note: If the initial value is not specified, the variable or collection is not reset to an empty string, but to the first value it received during the current call.
RECORDINGSCOPE ([value])	**recordingscope** ([value])	When used without an argument, returns the current value of the corresponding dialog context setting (*nomatch*, *recognition*, or *all*). When used with an argument, it sets the recording scope dialog context setting.
RECORDUTTERANCES ([value])	**recordutterances** ([value])	When used without an argument, returns the current value of the corresponding dialog context setting (*true* or *false*). When used with an argument, it sets the record utterances dialog context setting.
RESETVISITCOUNTER (referenceID)	**resetvisitcounter** (referenceID)	Resets the visit counter for the object with the specified reference ID. When *all* is given as argument, the visit counters for all objects are reset.
RETURNCODE ()	**returncode** ()	Returns the HTTP response code for the most recent HTTP or Web Service connector call.
SERVERCONTEXTMAPPINGURL ( )	**servercontextmappingurl** ( )	Returns the complete URL string for the context mapping of the corresponding server instance (e.g. *http://localhost:8099/VoiceObjects/*).
SERVERHOSTNAME ( )	**serverhostname** ( )	Returns the host name of the corresponding server instance.
SERVERINSTANCENAME ()	**serverinstancename** ()	Returns the name of the corresponding server instance.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions		
Name and Arguments		**Description**
Desktop	**XML**	
SERVERIP ([servletEngine])	**serverip** ([servletEngine])	Returns the IP address of the corresponding server instance. If the optional parameter *servletEngine* is set to *true* (the default), the address is retrieved from the servlet engine. If it is *false*, it is retrieved from the server configuration.
SERVERNAME ( )	**servername** ( )	Returns the logical name of the corresponding VoiceObjects server.
SERVERPORT ([servletEngine])	**serverport** ([servletEngine])	Returns the port of the corresponding server instance. If the optional parameter *servletEngine* is set to *true* (the default), the port is retrieved from the servlet engine. If it is *false*, it is retrieved from the server configuration.
SERVERSERVLET CONTEXT ( )	**serverservletcontext** ( )	Returns the servlet context of the corresponding server instance (e.g. */VoiceObjects*).
SERVERURL ([servletEngine])	**serverurl** ([servletEngine])	Returns the URL leading to the corresponding server instance (e.g. *http://172.22.23.45:8099/VoiceObjects/DialogMapping*). If the optional parameter *servletEngine* is set to *true* (the default), IP and port are retrieved from the servlet engine. If it is *false*, the corresponding values are retrieved from the server configuration.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions		
Name and Arguments		**Description**
Desktop	**XML**	
SESSION (varName, [value])	**session** (varName, [value])	Returns or sets information from the session context. The following arguments are supported: • *ANI* Calling number; same as ANI() • *DNIS* Called number; same as DNIS() • *RDNIS* Redirecting number; not available on all media platforms • *MPSID* Media platform session ID; not available on all media platforms • *AAI* Application to Application Information; not available on all media platforms • *GCID* Global call ID; not available on all media platforms • *UA* User-agent information; only available for Mobile Web driver • *CRMID* Unique caller or customer ID; can be used to join Infostore data with the CRM data warehouse • *customShadow* Custom data transferred from the media platform (optional). Availability of the information depends on the media platform. In addition, custom arguments can be used.
TRANSFERDESTINATION ()	**transferdestination** ()	Sets the transfer destination for Infostore. This is particularly useful when transfers are performed using Plug-In objects or Service Chaining.
VIDEOTYPE ([type])	**videotype** ([type])	When used without an argument, returns the code of the currently active video type (one of *video*, *videoaudio*, *none*). When used with an argument, it sets the currently active video type to this value. The argument must be one of the video type codes listed above. Note that the underlying media platform must be configured appropriately to support the video type that is being set.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions		
Name and Arguments		**Description**
Desktop	**XML**	
VISITCOUNTER (objRefID)	**visitcounter** (objRefID)	Returns the number of times the object with the specified reference ID has been visited within the current dialog.
VSN ()	**vsn** ()	Returns the name of the VoiceObjects service associated with the current dialog.

Miscellaneous Functions		
Name and Arguments		**Description**
Desktop	**XML**	
CONDITIONAL (cond, truevalue, falsevalue)	**conditional** (cond, truevale, falsevalue)	If *cond* is *true*, returns *truevalue*. Otherwise, returns *falsevalue*. *Cond* can be a Variable, Expression, Script, or Collection object, or a layer condition.

Copyright © 2001-2011 Voxeo Germany GmbH

Format

Overview

The Format object provides the concept of output formatting for dynamic content contained in Variable, Expression, Script, Collection, and Layer objects, which are presented to the caller in an Output object. It offers various formatting capabilities for voice and video applications based on text-to-speech (TTS) synthesis as well as text-to-audio (TTA) processing using prerecorded audio files.
Custom algorithms, also for formatting in video, text and Web applications, can be added using the *Formatting bus*. Examples are custom phone number or date formatting, or account number formatting in a banking application. For more information on how to embed custom formatting algorithms to a VoiceObjects installation, refer to Appendix B – *How to Use the Formatting Bus* in the *Administration Guide*.

The Object Definition below covers the configuration of the Format object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML, refer to the VoiceObjectsXML Definition paragraph.

The Format object belongs to the object category Resources.

Dialog Flow Scenario

The following dialog flow example as part of a flight information service demonstrates an Output object containing two prerecorded Audio snippets and two Variable objects to announce the terminal and gate information for a particular flight. The two Variable objects containing the terminal and gate data are played back with text-to-audio processing.

Object			Dialog Flow
			[*Your flight will leave on time out of terminal:*]
			[*B*]
			[*gate:*]
			[*fifty five*]

The second dialog flow example uses an Output object to read back a phone number (123 456 7890), which has been provided by the caller in an earlier dialog step and is stored in a Variable object. The assigned Format object reads back the phone number to the caller digit by digit in four blocks with a short pause in between, using text-to-speech synthesis.

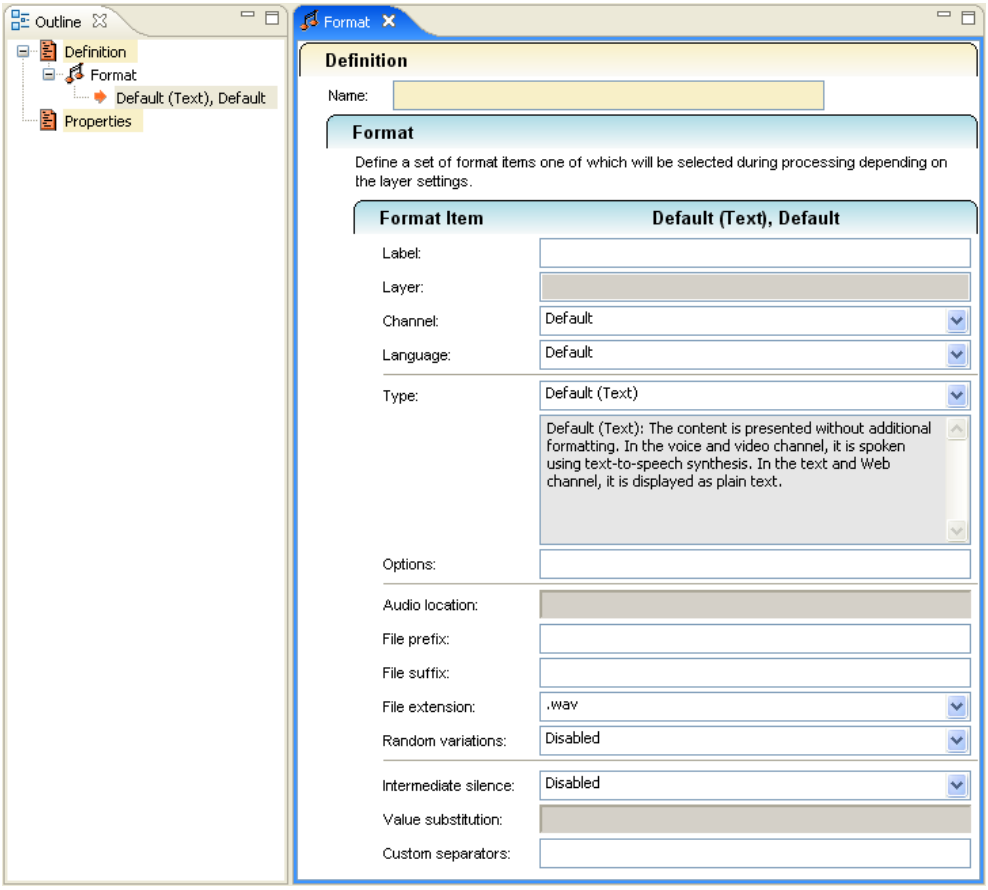
Copyright © 2001-2011 Voxeo Germany GmbH

Object			Dialog Flow
	Text		*I have the following phone number:*
			*one two three* **<pause>** *four five six* **<pause>** *seven eight* **<pause>** *nine zero*

Object Definition

The **Definition** of the Format object provides one section:

- Format



For further details regarding additional object configuration, refer to *Properties* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Format

In the **Format** section, define how the formatted content is played back within an Output object. To be able to specify different formatting depending on channel and potentially language, define separate **Format Items** and set the layers correspondingly, i.e. **Language**, **Channel**, and **Layer**.

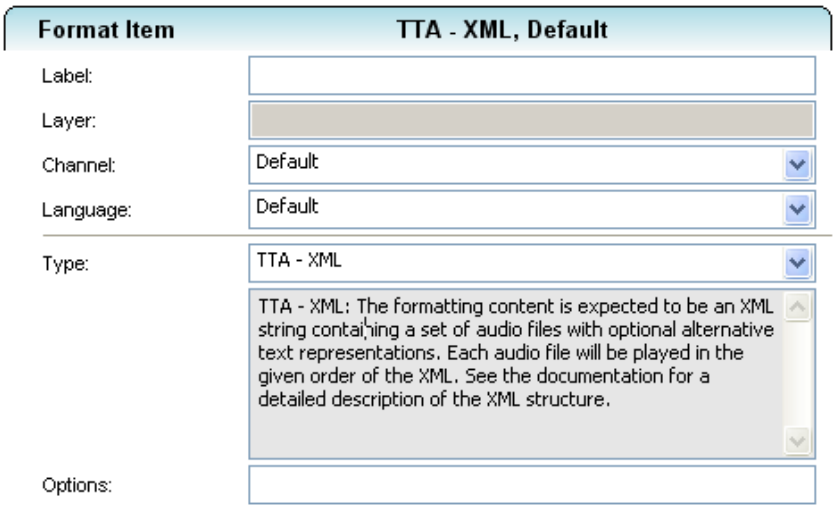
The Format object natively supports formatting for the voice channel only. Two different categories of formatting types are available:

- Text-To-Speech – (TTS) means that the media platform will process the formatted content with text-to-speech synthesis.
- Text-To-Audio – (TTA) means that the server will transform the formatted content into a sequence of references to audio files.

Note: Using the special formatting type *TTA-XML* you can define your own markup to be rendered by the server. This way you could instruct the server to also render corresponding formatting code for the video, text or Web channel, e.g. a `<video>` element for video applications, or a `<b>` element for Web applications.

The drop-down list of the **Type** field offers various choices for both formatting categories. Each list entry has either the prefix TTA or TTS. The default formatting type is set to plain text. In the voice and video channel, the text is played back through TTS; in the text and Web channel, the text is displayed as is, with no special formatting.

A description of the selected format type is shown in a text field right below the **Type** field.



Refer to Text-To-Speech Formatting Types and Text-To-Audio Formatting Types for further details on the provided algorithms.

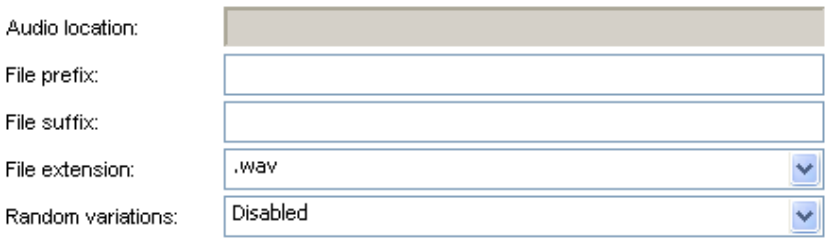
The optional **Options** field provides more precise TTS formatting for content types that can be expressed in diverse formats (e.g. date information). For example when using the formatting type *TTS – Date* the options string *dmy* specifies that the relevant date value will be presented in the order day, month and year. Refer to Formatting options for further details.

In case the object to be formatted is a Variable object and the type is one of the predefined TTA types, the *pronunciation value* of the variable will be taken as its

Copyright © 2001-2011 Voxeo Germany GmbH

value, not the value itself. Only if no pronunciation pattern is defined for the variable is the value taken. For more information on associating pronunciation patterns with a Variable object, refer to Input in this *Object Reference*.

The following five fields are only required when using a TTA formatting type to specify how and where the relevant audio files are stored and can be fetched from at call time.



The **Audio location** specifies the location of the relevant audio files (using a Resource Locator object).

The **File prefix** specifies a prefix that is applied to the filename. The file prefix can also be provided dynamically during call time by assigning a Variable, Expression, Layer, or Script object.

The **File suffix** specifies a suffix that is applied to the filename. The file suffix can also be provided dynamically during call time by assigning a Variable, Expression, Layer, or Script object.

The **Random variations** defines whether randomizing is enabled for the audio files. To enable randomizing, select a number from the drop-down list. If randomizing is enabled and multiple audio files are provided with the specified filename extended by different numbers, the server will randomly pick one of the available files with a random file extension between 1 and the selected number whenever the Audio object is used. True randomizing ensures that all available audio files are played before a repetition occurs. By default, the field is set to *Disabled* indicating that no randomizing is used. For more information on randomizing see the Output object in this *Object Reference*.

The **File extension** specifies the audio file extension. Instead of selecting one of the provided file extensions from the drop-down list, the extension can also be supplied by using a Variable, Expression, Layer, or Script object.

The complete filename for each audio file is generated at call time according to the following formula:

```
[prefix] + file + [suffix] + [random number] + [extension]
```

The specified TTA algorithm generates the parameter `file` in the formula.

The three fields at the bottom of the editor specify the following properties:



Copyright © 2001-2011 Voxeo Germany GmbH

The **Intermediate silence** field lets you play back a silence phase of a specified duration between each audio file when using TTA processing. For example when playing back a lengthy confirmation number a silence phase of 250ms can be added in between each digit.
The drop-down list offers a set of predefined silence timeframes. Alternatively a Variable, Expression, Layer, or Script object can be used to specify the silence phase at call time. The value needs to be numeric and will be interpreted in seconds.

The optional **Value substitution** field lets you specify a Collection object, which substitutes the formatted value with another value before presenting it to the caller. This mechanism can be used to transform an internal code or abbreviation into a complete name that makes sense to the caller, e.g. to substitute the state abbreviation *CA* with the full name *California*. Value substitution can be used for all TTS and TTA formatting types, including *Default (Text)*. In both cases, the substitution is performed first and then subsequent formatting is applied to the resulting substituted value. Refer to Value substitution collection for further details.

Finally the **Custom separators** field is used to overwrite the default set of separator tokens, which are used to split the formatted content into snippets (e.g. words and/or numbers when using TTA – Words/Numbers). The list of custom separator tokens needs to be concatenated without any separating character, e.g. use "_," to split the formatted content at each occurrence of underscore and comma. The default set of separator tokens contains: White space, dot, comma, semicolon, colon, dash, underscore, forward slash and backslash.

Text-To-Speech Formatting Types

The formatting type TTS without any specific instruction is the standard formatting setting. There are several choices available if the formatted content is known at design time (e.g. a time designation) and can therefore be mapped onto one of the listed formatting capabilities for the most optimal pronunciation and quality using text-to-speech synthesis:

- **TTS – Acronym** The formatted value is an acronym like *IBM* and is spoken as *eye bee em*.
- **TTS – Currency Amount** The formatted value is a currency amount. For example *$22* is spoken as *twenty two dollars*.
- **TTS – Date** The formatted value is a partial or complete date. For example the value *10/2001* is spoken as *October, two thousand and one* when using the TTS format *my*. The supported date formats indicate the form of the value and are usually *dmy, mdy*, *ymd*, *ym*, *my*, *md*, *y*, *m* and *d*. The desired output format can be specified in the **Options** field (see further below).
- **TTS – Digits** The formatted value object is a sequence of digits. For example *12* is spoken as *one two*.
- **TTS – Duration** The formatted value is a temporal duration. For example the value *10* is spoken as *ten hours* when using the TTS format *h*. The supported duration formats indicate the form of the value and are usually *hms*, *hm*, *ms*, *h*, *m* and *s*. The desired output format can be specified in the **Options** field (see further below).
- **TTS – E-mail Address** The formatted value is an Internet e-mail address. For example *name@company.com* is spoken as *name at company dot com*.
- **TTS – Literal** The formatted value is a sequence of characters. For example *Smith* is spoken as individual characters *ess em eye tea aitch*.
- **TTS – Measurement** The formatted value is a measurement. For example *10ft* is spoken as *ten feet*.

Copyright © 2001-2011 Voxeo Germany GmbH

- **TTS – Name** The formatted value is a name. For example *VoiceObjects* is spoken as *voice objects*.
- **TTS – Cardinal Number** The formatted value is a cardinal number. For example *12* is spoken as *twelve*.
- **TTS – Ordinal Number** The formatted value is an ordinal number. For example *12* is spoken as *twelfth*.
- **TTS – Postal Address** The formatted value is a text describing a postal address with street, city, state, and zip code separated by commas. For example *8080 Leesburg Pike, Reston, VA, 20190* is spoken as *eighty eighty <pause> Leesburg Pike <pause> Reston <pause> Virginia <pause> two zero one nine zero*.
- **TTS – Telephone Number** The formatted value is a telephone number. For example the phone number *123 456 7890* is spoken as *one two three <pause> four five six <pause> seven eight <pause> nine zero*.
- **TTS – Time** The formatted value is a partial or complete time designation. For example the value *8* is spoken as *eight o'clock* when using the TTS format *h*. The supported time formats indicate the form of the value and are usually *hms*, *hm* and *h*. The desired output format can be specified in the **Options** field (see further below).
- **TTS – URI** The formatted value is an Internet identifier such as *www.company.com*. It will be spoken out as *double u double u double u dot company dot com*.

Note: Some of the certified media platforms may not support all of the listed TTS types. The support can also differ depending on the integrated TTS engine or on the language used. Further details can be found in the corresponding platform documentation. A few of the certified media platforms support various additional TTS formatting capabilities, e.g.:

- **Airport** The formatted value is an airport code. For example *CGN* is spoken as *Cologne*.
- **Airline** The formatted value is an airline code. For example *LH* is spoken as *Lufthansa*.
- **Equity** The formatted value is an equity symbol. For example *CSCO* is spoken as *Cisco Systems*.

The output type can also be defined by referencing a Variable, Expression, Layer, or Script object instead of selecting a predefined formatting type. The provided string will be passed into the `<say-as>` element of the generated markup code (e.g. VoiceXML). Here is an example of providing the string `MyOutputType`, depending on the supported VoiceXML standard as defined in the media platform driver of VoiceObjects Server:

- **`<say-as>`** element with the **class** attribute (VoiceXML 1.0 Specification - March 2000)

```
<sayas class="MyOutputType">
    [Formatted value]
</sayas>
```

Copyright © 2001-2011 Voxeo Germany GmbH

- **<say-as>** element with the **type** attribute (VoiceXML 2.0 Specification - October 2001)

```
<say-as type="MyOutputType">
    [Formatted value]
</say-as>
```

- **<say-as>** element with the **interpret-as** attribute (VoiceXML 2.0 Specification - March 2004)

```
<say-as interpret-as="MyOutputType">
    [Formatted value]
</say-as>
```

If required, an additional TTS options string can be supplied (refer to Formatting options below). This string is usually concatenated with the formatting type, using a colon in between. For example, the TTS options string `MyFormat` together with the formatting type string `MyOutputType` will be rendered in the following way:

- **<say-as>** element with the **class** attribute (VoiceXML 1.0 Specification - March 2000)

```
<sayas class="MyOutputType:MyFormat">
    [Formatted value]
</sayas>
```

- **<say-as>** element with the **type** attribute (VoiceXML 2.0 Specification - October 2001)

```
<say-as type="MyOutputType:MyFormat">
    [Formatted value]
</say-as>
```

- **<say-as>** element with the **interpret-as** attribute but not supporting the **format** attribute (VoiceXML 2.0 Specification - March 2004)

```
<say-as interpret-
as="MyOutputType:MyFormat">
    [Formatted value]
</say-as>
```

When the media platform supports the **format** attribute within the VoiceXML `<say-as>` element, the TTS options string will be assigned to the format attribute and not concatenated with the string for the output type.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<say-as>** element with the **interpret-as** attribute and support for the **format** attribute (VoiceXML 2.0 Specification - March 2004)

```
<say-as interpret-as="MyOutputType"
        format="MyFormat">
    [Formatted value]
</say-as>
```

Formatting options

The following three formatting types offer additional formatting options:

TTS – Date
Date information can be provided as a partial date (e.g. only year and month) or as a complete date (containing year, month, and day). In either case the TTS processor needs to know what to do with each part of the information to produce a suitable output to the caller. For example the date information *01/02/03* is ambiguous. By giving the TTS processor the formatting hint *dmy*, which translates into day, month, and year, the date can be synthesized to *first of February two thousand and three*. The following formatting options are provided when using a date designation:

- **dmy** day, month and year
- **mdy** month, day and year
- **ymd** year, month and day
- **ym** year and month
- **my** month and year
- **md** month and day
- **y** year only
- **m** month only
- **d** day only

TTS – Duration
If the formatted content is a duration value, it can be provided in several ways such as the number of hours *(5)* or the number of minutes and seconds *(6:21)*. By giving the TTS processor the formatting hint *ms*, which translates into minutes and seconds, the duration *6:21* can be synthesized to *six minutes and twenty-one seconds*. When providing the formatting hint *hm*, it is synthesized to *six hours and twenty-one minutes*. The following formatting options are provided when using a duration value:

- **hms** hours, minutes and seconds
- **hm** hours and minutes
- **ms** minutes and seconds
- **h** hours only
- **m** minutes only
- **s** seconds only

Copyright © 2001-2011 Voxeo Germany GmbH

TTS – Time
A time value can also be provided as partial or complete information. The following formatting options are available when using a time designation:

- **hms** hours, minutes and seconds
- **hm** hours and minutes
- **h** hours only

The **Options** field can either be specified as a constant string or via the value of a Variable, Expression, Layer, or Script object, which gets evaluated at call time.

Note: In all cases the content of the **Options** field should be provided in lower case notation.

Text-To-Audio Formatting Types

Text-to-audio (TTA) is an integrated formatting mechanism of VoiceObjects Server that converts data in its raw format to a sequence of references to prerecorded audio resources. The formatting type TTA provides five different ways to convert the formatted data content:

- TTA - Literals/Digits The formatting content is a sequence of characters and/or digits. Each character or digit is played via a separate audio file. The audio files are named with the corresponding character or digit (e.g. *a.wav* or *1.wav*).
- TTA - Literals/Digits (ASCII) Each literal or digit of the formatted content is played back to the caller as a separate audio resource. The audio files are named with the corresponding 3-digit decimal ASCII code (e.g. *097.wav* for character *a* or *049.wav* for digit *1*).
- TTA - Words/Numbers The formatted content is chopped into words and number blocks where each of them is played back as a separate audio resource. The files are named according to word or number block (e.g. *airbus.wav* or *380.wav*).
- TTA - Complete Value The complete formatted content is played back as a single audio resource. The file is named according to content itself where white spaces are replaced with an underscore (e.g. *airbus_a_380.wav*).
- TTA - Files The formatted content is expected to be a single audio file or a white space separated list of audio files (e.g. *airbus_a_380.vox boeing_747.vox boeing_777.vox*).
- TTA - XML The formatted content is expected to be an XML string containing a set of audio files with optional alternative text representations. Each audio resource will be played to the caller in the order of the XML structure.
  This special formatting type also allows defining markup to be rendered, so it can also be used to define formatting for the video, text and Web channel.

Tip: You can create your own formatting algorithms and embed them in VoiceObjects Desktop and VoiceObjects Server. See Appendix B – *How to Use the Formatting Bus* in the *Administration Guide* for more information on how to do this.

If you are using sites and have setup a site prefix (*Resource Locator URL*) for external resources in your installation, this prefix will be applied to all TTA types. Custom algorithms embedded through the Formatting bus will not use the site prefix, though, unless the Format object defines an audio location. For more information on the site settings, refer to Chapter 3 – *User Management - Managing Sites* in the *Administration Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

Note: All audio files for TTA processing must be of a single audio type. Mixing wav and vox files in one playback sequence, for example, is not supported.

The TTA processing uses the rendered audio filename as the alternative text definition for TTS processing. If a missing audio file is requested by the media platform the corresponding filename will be played back via TTS. When using the TTA functionality it is very helpful during the development phase to specify a maintainer e-mail address in the corresponding Service object. In this case the media platform will send an e-mail to this address indicating the missing audio file.

In case of Variable objects, all predefined TTA algorithms use the *pronunciation value* of the variable, not the real value. Only if no pronunciation pattern is defined for the variable is the value taken. For more information on this, refer to Input in this *Object Reference*.

TTA – Literals/Digits

The following rules are applied when generating audio filenames character by character or digit by digit:

- Variable content is converted to lower case representation.
- Leading and trailing white spaces are removed.
- All white spaces and special characters not in [a-z] or [0-9] are removed.
- For each literal or digit a corresponding filename is generated according to the following formula:

```
[prefix] + [literal/digit] + [suffix] + [extension]
```

Examples for generated filename sequences are:

Formatted Content	Prefix	Suffix	Extension	Filename Sequence
Hello world!			.wav	h.wav e.wav l.wav l.wav o.wav w.wav o.wav r.wav l.wav d.wav
- Hello world! -	8kHz_		.wav	8kHz_h.wav 8kHz_e.wav 8kHz_l.wav 8kHz_l.wav 8kHz_o.wav 8kHz_w.wav 8kHz_o.wav 8kHz_r.wav 8kHz_l.wav 8kHz_d.wav

Copyright © 2001-2011 Voxeo Germany GmbH

Formatted Content	Prefix	Suffix	Extension	Filename Sequence
S F O			.vox	s.vox f.vox o.vox
Müller			.wav	m.wav l.wav l.wav e.wav r.wav
(0)171-549-0087		en	.wav	0en.wav 1en.wav 7en.wav 1en.wav 5en.wav 4en.wav 9en.wav 0en.wav 0en.wav 8en.wav 7en.wav
+49 2205 845 100	de/		.wav	de/4.wav de/9.wav de/2.wav de/2.wav de/0.wav de/5.wav de/8.wav de/4.wav de/5.wav de/1.wav de/0.wav de/0.wav

Alternative text for each individual audio file is the corresponding original literal or digit.

Use cases for this processing include:

- Phone number
- Credit card number
- Account number
- Customer ID
- PIN
- Zip code
- Social security number
- Digits
- Spelling

Copyright © 2001-2011 Voxeo Germany GmbH

TTA – Literals/Digits (ASCII)

The following rules are applied when generating audio filenames literal by literal or digit by digit by converting them to ASCII decimal code (3-digits):

- Variable content is converted to lower case representation.
- All white spaces are removed.
- For each literal or digit a corresponding filename is generated according to the following formula:

```
[prefix] + [ASCII code] + [suffix] + [extension]
```

Examples for generated filename sequences are:

Formatted Content	Prefix	Suffix	Extension	Filename Sequence
Hello world!			.wav	104.wav [h] 101.wav [e] 108.wav [l] 108.wav [l] 111.wav [o] 119.wav [w] 111.wav [o] 114.wav [r] 108.wav [l] 100.wav [d] 033.wav [!]
S F O	ascii_			ascii_115 [s] ascii_102 [f] ascii_111 [o]
Müller			.wav	109.wav [m] 252.wav [ü] 108.wav [l] 108.wav [l] 101.wav [e] 114.wav [r]
+49-(0)171-5490087		en	.wav	043en.wav [+] 052en.wav [4] 057en.wav [9] 045en.wav [-] 040en.wav [(] 048en.wav [0] 041en.wav [)] 049en.wav [1] 055en.wav [7] 049en.wav [1] 045en.wav [-] 053en.wav [5] 052en.wav [4] 057en.wav [9] 048en.wav [0] 048en.wav [0] 056en.wav [8] 055en.wav [7]

Copyright © 2001-2011 Voxeo Germany GmbH

Alternative text for each individual audio file is the corresponding original literal or digit.

Use cases for this processing include:

- Phone number
- Credit card number
- Account number
- Customer ID
- PIN
- Zip code
- Social security number
- Digits
- Spelling

TTA – Words/Numbers

The following rules are applied when generating audio filenames for each single word or number:

- Variable content is converted to lower case representation.
- Leading and trailing white spaces are removed.
- White space, dot, comma, semicolon, colon, dash, underscore, forward slash and back slash are used by default as separators to split the content into words and numbers. The set of separators can be customized using the **Custom separators** field.
- All remaining special characters not in [a-z] or [0-9] are removed.
- For each word or number a corresponding filename is generated according to the following formula:

```
[prefix] + [word / number] + [suffix] + [extension]
```

Examples for generated filename sequences are:

Formatted Content	Prefix	Suffix	Extension	Filename Sequence
Hello world!			.wav	hello.wav world.wav
“ Hello world! ”		_DE	.mp3	hello_DE.mp3 world_DE.mp3
- Hello world! -	en/		.vox	en/hello.vox en/world.vox
gate D::47			.wav	gate.wav d.wav 47.wav

Copyright © 2001-2011 Voxeo Germany GmbH

Formatted Content	Prefix	Suffix	Extension	Filename Sequence
http://www.company.com	us.		.au	us.http.au us.www.au us.company.au us.com.au
“ Airbus: A-380 ”			.wav	airbus.wav a.wav 380.wav
Munich or Hamburg			.wav	munich.wav or.wav hamburg.wav
CA, FL, VA and TX			.wav	ca.wav fl.wav va.wav and.wav tx.wav
Joe, Tom or optionally Ben			.wav	joe.wav tom.wav or.wav optionally.wav ben.wav
(0)171-264-5597			.wav	0171.wav 264.wav 5597.wav
IP: 255.255.255.0			.wav	ip.wav 255.wav 255.wav 255.wav 0.wav

Alternative text for each individual audio file is the corresponding original word or number.

Use cases for this processing include:

- Person, company, product and property names
- Country, city and state names
- Flight gate or railway track number
- All kind of codes (state, equity, currency, airline, airport, etc.)
- Predefined set of Internet addresses
- Predefined set of phone numbers

TTA – Complete Value

The following rules are applied when generating audio filenames for the complete content:

- Variable content is converted to lower case representation.

Copyright © 2001-2011 Voxeo Germany GmbH

- Leading and trailing white spaces are removed.
- White space, dot, comma, semicolon, colon, dash, underscore, forward slash and back slash are used by default as separators to split the content into words and numbers. The set of separators can be customized using the **Custom separators** field.
- All remaining special characters not in [a-z] or [0-9] are removed.
- All separators are replaced by an underscore, where multiple separators are replaced by a single underscore.
- A corresponding filename is generated according to the following formula:

```
[prefix] + [modified content] + [suffix] + [extension]
```

Examples for generated filename sequences are:

Formatted Content	Prefix	Suffix	Extension	Filename Sequence
Hello world!	8kHz_		.wav	8kHz_hello_world.wav
“ Hello world! ”		_DE	.mp3	hello_world_DE.mp3
- Hello world! -	en/		.vox	en/hello_world.vox
New York City			.wav	new_york_city.wav
gate D::47			.wav	gate_d_47.wav
Airbus: A-380			.wav	airbus_a_380.wav
S F O	us.		.au	us.s_f_o.au
Müller			.wav	mller.wav

Alternative text for the audio file is the original unmodified content.

Use cases for this processing include:

- Person, company, product and property names
- Country, city and state names
- Flight gate or railway track number
- All kinds of codes (state, equity, currency, airline, airport, etc.)

TTA – Files

The following rules are applied when the content already provides the reference of one or multiple audio files:

- Leading and trailing white spaces are removed.

Copyright © 2001-2011 Voxeo Germany GmbH

- White space is used as separator to split the content into separate filenames or references.
- Each filename is processed as provided. Any specified TTA prefix, suffix or extension settings are ignored. Only the TTA locator (if provided) is used to generate each complete file reference.

Examples for generated filename sequences are:

Formatted Content	Filename Sequence
Hello.wav	Hello.wav
Hello-en-US.wav World-en-US.wav	Hello-en-US.wav World-en-US.wav
1.vox 7.vox 15.vox and.vox 22.vox	1.vox 7.vox 15.vox and.vox 22.vox
de/CA.wav de/or.wav de/VA.wav	de/CA.wav de/or.wav de/VA.wav
http://255.255.255.0/jingle.mp3	http://255.255.255.0/jingle.mp3
file://MusicBox/track-14.v1.au file://MusicBox/track-28.v3.au file://MusicBox/track-43.v2.au file://MusicBox/track-81.v1.au file://MusicBox/track-92.v1.au	file://MusicBox/track-14.v1.au file://MusicBox/track-28.v3.au file://MusicBox/track-43.v2.au file://MusicBox/track-81.v1.au file://MusicBox/track-92.v1.au

Alternative text for each individual audio file is the corresponding filename.

Use cases for this processing include:

- Ring tones
- Voice mail messages
- Music tracks
- All kinds of prerecorded news (business, weather, traffic, sports, etc.)
- Dating personals
- Recordings

TTA – XML

When using TTA-XML, a custom sequence of audio files with optional alternative texts can be provided. Alternatively, custom markup code can be provided that is used as-is inside the code rendered by the server. This option can be used to play video instead of audio, or to make use of specific formatting capabilities provided by certain media platforms or browsers. It can also be used to embed Web formatting, like `<b>` or `<center>`, into the XHTML code produced by the server in the Web channel.
The XML format is as shown below.

Copyright © 2001-2011 Voxeo Germany GmbH

Note: When using special characters as part of the filename or the alternative text definitions the provided content must be enclosed in CDATA elements.

```
<root>
  <row>
    <col name="markup"><![CDATA[Markup code]]></col>
    <col name="objectReference"><![CDATA[Reference ID]]></col>
    <col name="file"><![CDATA[File name]]></col>
    <col name="text"><![CDATA[Alternative text]]></col>
    <col name="status"><![CDATA[Optional message]]></col>  </row>
  . . .
</root>
```

Any number of rows may be provided within the `<root>` element. Each row may contain a column named “markup”, a column named “objectReference”, or columns named “file” and (optionally) “text”.
In either case it is also allowed to provide a column named “status”, which contains a text message that is written to the service's error log file (see Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide* for more details). It may be used to indicate error conditions, e.g. a wrongly formatted input string.

If a column named “markup” is provided, then its content is embedded into the code generated by the server. No processing or checking takes place on the markup code that is provided, so broken code inside a “markup” column may lead to invalid code being sent to the media platform or browser.

By using "markup" in the voice or video channel, proprietary code can be embedded into the rendered VoiceXML. Media platform vendors such as Genesys, Intervoice, Nortel, and Nuance, for instance, provide their own TTA-like algorithms to use through proprietary VoiceXML element extensions of `<value>` or `<audio>` (see, e.g., http://cafe.bevocal.com/docs/vxml/voices.html#279348). Another scenario is to embed video elements in this column, to play back content through a concatenation of video files.

If a column named “objectReference” is provided, its content is interpreted as the Reference ID of an object. This object is evaluated and the resulting markup code is inserted. Note that the referenced object must be one of Output, Audio, Video, Silence, Script, Variable, Collection, Expression, or Layer. Attempts to reference an object that cannot be evaluated lead to an *Error-Internal*. If the referenced object cannot be found, an error is logged but processing continues.

If a column named “file” and optionally a column named “text” is provided, then the following rule will be applied to the XML string containing files and optional alternative text representations:

- Each filename and alternative text is processed as provided. Any specified TTA prefix, suffix or extension settings are ignored. Only the TTA locator (if provided) is used to generate each complete file reference.

Note that the “file” column, when present, must not be empty, and that a “text” column may only be used in addition to a “file” column to define alternative text for that file.

Examples for generated filename sequences are:

Copyright © 2001-2011 Voxeo Germany GmbH

Formatted Content	Filename Sequence
<root> <row> <col name="file">hello.wav</col> <col name="text">hello</col> </row> <row> <col name="file">world.wav</col> <col name="text">world</col> </row> </root>	hello.wav world.wav
<root> <row> <col name="file">A340.wav</col> <col name="text">Airbus A 3 40</col> </row> <row> <col name="file">A380.wav</col> <col name="text">Airbus A 3 80</col> </row> <row> <col name="file">B747.wav</col> <col name="text">Boeing 7 4 7</col> </row> <row> <col name="file">B777.wav</col> <col name="text">Boeing Triple 7</col> </row> </root>	A340.wav A380.wav B747.wav B777.wav
<root> <row> <col name="file">1.vox</col> </row> <row> <col name="file">7.vox</col> </row> <row> <col name="file">15.vox</col> </row> <row> <col name="file">and.vox</col> </row> <row> <col name="file">22.vox</col> </row> </root>	1.vox 7.vox 15.vox and.vox 22.vox

Copyright © 2001-2011 Voxeo Germany GmbH

Formatted Content	Filename Sequence
<root> <row> <col name="file">de/CA.au</col> <col name="text">Kalifornien</col> </row> <row> <col name="file">de/and.au</col> <col name="text">und</col> </row> <row> <col name="file">de/TX.au</col> <col name="text">Texas</col> </row> </root>	de/CA.au de/and.au de/TX.au

Examples for native markup code are:

Formatted Content
Nuance Cafe – Recorded Voices Extension <root> <row> <col name="markup"> <![CDATA[<say-as type="equity" bevocal:mode="recorded"> <value expr="'IBM'" /> </say-as>]]> </col> </row> </root>
Genesys – Playing Dynamic Data Extension <root> <row> <col name="markup"> <![CDATA[<script src="Languages/en-US/PlayBuiltinType.js" /> <audio expr="PlayBuiltinType("4532.99", "currency");" xmlns="http://www.telera.com/vxml/2.0/ext/20020430" />]]> </col> </row> </root>

Copyright © 2001-2011 Voxeo Germany GmbH

Formatted Content

Intervoice – ISay Extension

```
<root>
  <row>
    <col name="markup">
      <![CDATA[
      <value expr="'3333'"
             mode="recorded"
             class="digits"
             xml:lang="en-US" />
      ]]>
    </col>
  </row>
</root>
```

Nortel – MMF Prompt Extension

```
<root>
  <row>
    <col name="markup">
      <![CDATA[
      <value expr="'20060331'"
             mode="recorded"
             class="date:ymd" />
      ]]>
    </col>
  </row>
</root>
```

Value substitution collection

The mechanism of value substitution is provided on the basis of a Collection object, which is used as a substitution dictionary. The XML format of the collection requires a column with the name *key* that defines the lookup key, which needs to map the formatted value. A column *default* can be provided to specify the corresponding substitution values. Alternatively (or in addition), multiple columns with language codes such as *en-UK* or *fr-FR* can be added to define the corresponding substitution values in a localized version.

The first example provides a substitution collection to replace all US state abbreviations with the full name. This example is using the column name *default* to define the substitution values:

```
<root>
  <row>
    <col name="key">AK</col>
    <col name="default">Alaska</col>
  </row>
  <row>
    <col name="key">AL</col>
    <col name="default">Alabama</col>
  </row>
  . . .
  <row>
    <col name="key">WI</col>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
    <col name="default">Wisconsin</col>
  </row>
  <row>
    <col name="key">WY</col>
    <col name="default">Wyoming</col>
  </row>
</root>
```

The second example outlines a substitution collection containing the numerical representation of a month (1-12) with the localized name in US English (using the column name *en-US*) and German (using the column name *de-DE*):

```
<root>
  <row>
    <col name="key">1</col>
    <col name="en-US">January</col>
    <col name="de-DE">Januar</col>
  </row>
  <row>
    <col name="key">2</col>
    <col name="en-US">February</col>
    <col name="de-DE">Februar</col>
  </row>
  . . .
  <row>
    <col name="key">11</col>
    <col name="en-US">November</col>
    <col name="de-DE">November</col>
  </row>
  <row>
    <col name="key">12</col>
    <col name="en-US">December</col>
    <col name="de-DE">Dezember</col>
  </row>
</root>
```

Note: If a substitution collection is specified and the formatted value cannot be found in the *key* column, no substitution will take place. In this case the original value will be used when playing it back to the user via TTS or TTA processing. The same will happen when the dialog is processed in a particular language for which no column with the corresponding language code is defined.

VoiceObjectsXML Definition

The Format object is represented by the VoiceObjectsXML element `<format>`. It has no special attributes and one possible child `<formatItem>`.

In addition, the element has the standard attributes described in the *XDK Guide*.

Format

Children

- **`+<formatItem>`**
  Defines the list of available Format items.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<format name=”Digits”>
  <formatItem channel=”voice” type=”tts-digits”/>
  <formatItem channel=”textWeb” type=”tts”/>
</format>
```

FormatItem

Attributes

- **`label`**
  A text string providing a name for the Format item.
- **`language`**
  Defines the language for which this Format item is valid. Can be *default* or a valid language code (e.g. *de-DE*, *en-US*, etc.). If not specified, defaults to *default*. Appendix A – Language Codes contains a list of all language codes available in VoiceObjects together with the respective language they represent.
- **`channel`**
  Defines the channel(s) for which this Format item is valid. Can be *default*, *voice*, *video*, *text*, *web*, *voiceVideo*, or *textWeb*. If not specified, defaults to *default*.
- **`layer`**
  Defines the layer for the Format item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **`type`** [required]
  Defines the type of output formatting to be used. Legal values are *tts*, *tts-acronym*, *tts-cardinal*, *tts-currency*, *tts-date*, *tts-digits*, *tts-duration*, *tts-email*, *tts-literals*, *tts-measurement*, *tts-name*, *tts-ordinal*, *tts-address*, *tts-phone*, *tts-time*, *tts-uri*, *tta-complete*, *tta-words*, *tta-files*, *tta-xml*, *tta-literals*, *tta-ascii*. If not specified, defaults to *tts*.
  If you want to use a custom TTA algorithm, set the **`type`** attribute to the name of the desired algorithm, which is set in the TTA definition file.
- **`options`**
  Specifies options to be applied when processing the output formatting.
- **`audioLocation`**
  Defines the location from where audio files are to be retrieved. Must be a reference to a resource locator.
- **`prefix`**
  Defines a prefix that is attached to all audio files referenced in the formatted output. May be static text or a reference to a Variable, Expression, or Script object.
- **`suffix`**
  Defines a suffix that is attached to all audio files referenced in the formatted output. May be static text or a reference to a Variable, Expression, or Script object.
- **`random`**
  Indicates whether randomization should be used. Can be either *disabled* or an integer >= 2 specifying the number of versions in which the file is available. If not specified, defaults to *disabled*.

Copyright © 2001-2011 Voxeo Germany GmbH

- **`extension`**
  Specifies the file extension for audio files as either a constant value or a reference to a Variable, Expression, or Script object. Legal constant values are *none*, *wav*, *aif*, *aiff*, *dwd*, *mp3*, *snd*, *au*, *voc*, *vox*. If not specified, defaults to *wav*.
- **`silence`**
  Defines the duration of a pause between every two consecutive audio files. Either *disabled* or a numerical value interpreted as seconds. If not specified, defaults to *disabled*. May be static text or a reference to a Variable, Expression, or Script object.
  When importing an XDK application into the VoiceObjects Metadata Repository via VoiceObjects Desktop, this value must be one of those presented in the Desktop drop-down list (0.1, 0.25, 0.5, 0.75, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 70, 80, 90, 100, 110, 120), or an object reference.
- **`valueSubstitution`**
  Reference to a Collection object defining a substitution table. This substitution is applied before processing the output formatting.
- **`separators`**
  Defines a list of custom separators used when splitting data. May be static text or a reference to a Variable, Expression, or Script object.

Example

```
<format name=”Digits”>
  <formatItem channel=”voice” type=”tts-digits”/>
  <formatItem channel=”textWeb” type=”tts”/>
</format>
```

Object Interoperability

The Format object is compatible with the following other objects which can contain dynamic content:

Icon	Object Name	Use Case Example
	Collection	A Format object can be used within a Collection object to format the collection values.
	Expression	A Format object can be used within an Expression object to format the evaluated expression result.
	Script	A Format object can be used within a Script object to format the evaluated script result.
	Variable	A Format object can be used within a Variable object to format the variable value.
	Layer	A Format object can be used within a Layer object to format the active layer state(s).

Copyright © 2001-2011 Voxeo Germany GmbH

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Format object should be as short as possible and at the same time precise enough that other users will know immediately what the formatting definition is good for.

The table below is listing two examples as a general guideline:

Name	Description
TTS Phone Number	Say content as phone number with text-to-speech processing.
TTA Phone Number	Say content as phone number with text-to-audio processing. Silence of 250ms seconds is added between each phone number digit.

Copyright © 2001-2011 Voxeo Germany GmbH

Goto

Overview

The Goto object lets you abandon the current dialog flow and process a selected object instead. You can also process the object as a sub-dialog and then return to the original position in the dialog flow.

Note: The Goto object should be used sparingly and with caution. In the majority of cases, the intended dialog flow can equally be achieved by simply referencing the destination object directly.

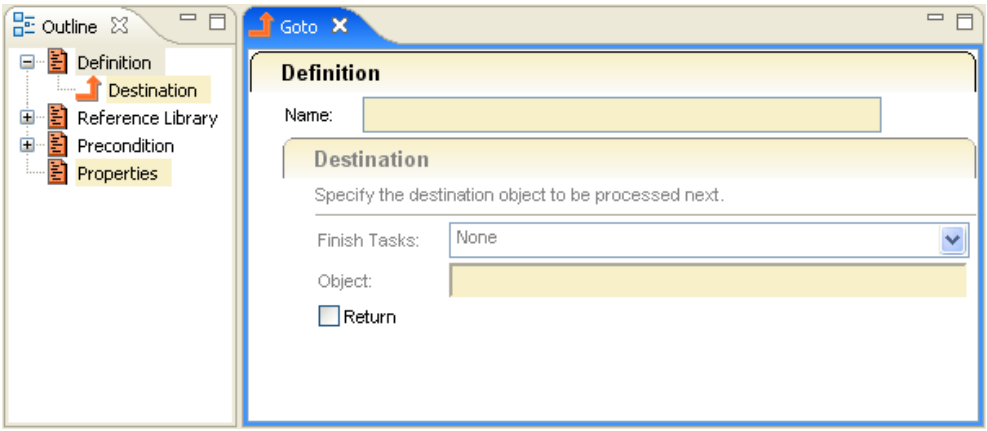
The Object Definition below covers the configuration of the Goto object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML, refer to the VoiceObjectsXML Definition paragraph.

The Goto object belongs to the object category Logic.

Object Definition

The **Definition** of the Goto object provides one section:

- Destination



In addition, the Goto object provides a **Reference Library** section for advanced use. The **Reference Library** section does not belong to the actual **Definition** section but is a separate major section and thus represented as a first level entry in the **Outline** (and as a separate tab in Desktop for Web).

For further details regarding additional object configuration refer to *Precondition* and *Properties* in this *Object Reference*.

Destination

In the **Object** field, the destination object to be processed next is defined.

The **Finish Tasks** field defines whether currently active business tasks are to be finished or not when the Goto object is processed. It can be set to *None* (the default), which means that no task will be finished, *Active*, which means that only the currently active task will be finished, or *All*, which means that all tasks, active or inactive, will be

Copyright © 2001-2011 Voxeo Germany GmbH

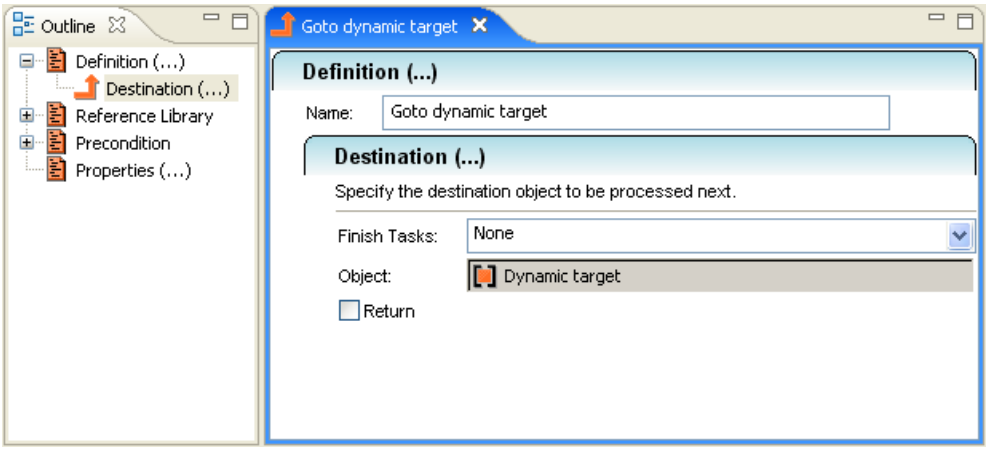
finished. In case one or more tasks get finished, they will automatically be marked with status *Caller Abort*, which means they will be logged as *incomplete* to Infostore.

If the **Return** check box is selected, the dialog processing returns to the current position in the active dialog flow after having processed the destination object.

If the **Return** check box is clear (the default), the currently active dialog flow is abandoned and the destination object is processed. Note that in this case, when the destination object (which might be a Module or a Sequence object, thus a whole sub-application) has been processed completely, the call does not return to the original flow but will simply terminate.

Reference Library

The Goto object offers the capability to process dynamic dialog blocks at call time by referencing a Variable object as the destination object. The content of the Variable object is interpreted as the reference ID of an object, and the corresponding object is processed.



Note: If no object with this reference ID is available, VoiceObjects Server triggers an *Error – Internal*.

When using this advanced Goto behavior, care must be taken that the objects that are to be dynamically processed are all included within the application definition cache that the server loads when deploying a service (see Chapter 4 – *Service Deployment* in the *Deployment Guide* for details). This is best ensured by placing all objects that may potentially be processed into the Goto object's **Reference Library**. You can also place these objects inside a Module object. See Module object description in this *Object Reference* for more details.

Copyright © 2001-2011 Voxeo Germany GmbH

VoiceObjectsXML Definition

The Goto object is represented by the VoiceObjectsXML element `<goto>`. It has three attributes and two groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

Goto

Attributes

- **`object`** [required]
  Defines the object to be processed.
- **`finishTasks`**
  Defines whether business tasks are to be finished or not. One of *none*, *active*, *all*. If not specified, defaults to *none*.
- **`return`**
  Defines the return mode. Can either be *true* or *false*. If not defined, it defaults to *false*.

Children

- **`<expression usage=”precondition”>`** or
  `<variable usage=”precondition”>` or
  `<collection usage=”precondition”>` or
  `<script usage=”precondition”>`
  Defines the precondition for the Goto object.
- **`<sequence usage=library>`**
  Defines the object library for the Goto object (to ensure that the target object is in the service cache when using dynamic targets).

Example

```
<goto object=”#MainMenu”/>
```

Object Interoperability

The following table contains all the object types that can reference a Goto object. Note that in the overwhelming majority of cases, the use of the Goto object can be replaced by simply using the destination object of the Goto itself.

Icon	Object Name	Use Case Example
	Module	A Goto object can be used within the embedded sequence of a Module object.
	Sequence	A Goto object can be used within a Sequence object in a dialog flow.
	Menu	A Goto object can be used as the destination object within a Menu item.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Object Name	Use Case Example
	Confirmation	A Goto object can be used as the destination object within a Correction item inside a Confirmation object.
	If	A Goto object can be either in the THEN item or the ELSE item of an If object.
	Case	Goto objects can be used in any WHEN item or the ELSE item of a Case object.
	Loop	Goto objects can be used within the embedded sequence of a Loop object.
	Goto	A Goto object can be referenced via a Goto object. Note that while this is technically possible, it normally does not make much sense.
	Hyperlink	A Goto object can be the destination of a Hyperlink object. Note: While being technically possible this should be avoided.
Any object type providing event handling		An object can be processed via a Goto object when a certain event occurs.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Goto object should describe the target of the Goto. The short description should contain a brief explanation of what is done within the target dialog flow. The table below lists two examples:

Name	Description
 Activate novice user dialog	After the third No Match event, the dialog is switched to a more guided version to help novice users.
 Return to main menu	Terminate the current dialog flow and jump back to the main menu. Used when the transaction could not be completed successfully.

Copyright © 2001-2011 Voxeo Germany GmbH

Grammar

Overview

The Grammar object defines a recognition grammar, which can be used in all objects that accept caller input (such as Input, Menu, or Confirmation objects). A recognition grammar defines the input a caller can respond with to a particular request prompt. In the voice and video channel, this can be a set of phrases that a caller might say or DTMF keys they might press on the phone's keypad. In the text and Web channel, the grammar can define a set of options; depending on if and how the grammar is displayed, the caller can either select an option, or respond with free text that can then be mapped to slot values by the server.

Tip: To develop dynamic speech or DTMF grammars effectively you can use a grammar tool like *NuGram IDE*, which provides a complete set of Eclipse tools to author, test, and debug speech or DTMF recognition grammars. For further details and a free download refer to the Nu Echo Website (www.grammarserver.com) or the VoiceObjects Developer Portal (http://developers.voiceobjects.com/partners/).

The Object Definition below covers the configuration of the Grammar object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML, refer to the VoiceObjectsXML Definition paragraph.

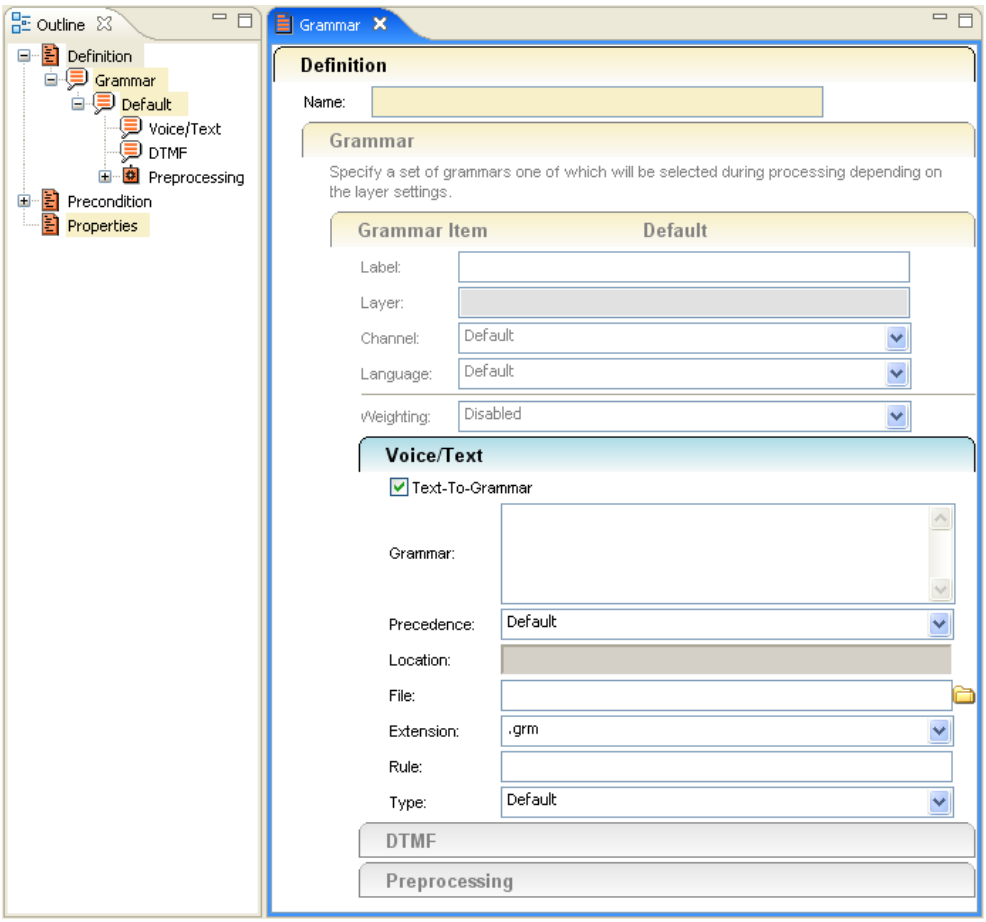
The Grammar object belongs to the object category Resources.

Object Definition

The **Definition** of the Grammar object provides one section:

- Grammar

Copyright © 2001-2011 Voxeo Germany GmbH



For further details regarding additional object configuration refer to Precondition and Properties in this *Object Reference*.

Grammar

In the **Grammar** section, one or more Grammar items can be defined.

Note: The structure of the **Grammar** editor in Desktop for Web is slightly different but the provided fields are actually the same.

Each **Grammar Item** section contains the following fields:

Property	Description
Label	Optional parameter to identify the Grammar item in a list.
Layer	Optional parameter to define a layer condition for the specific Grammar item. At call time, only Grammar items with layer conditions that evaluate to *true* are activated. For more information on layers, see the Layer object in this *Object Reference*, or Chapter 7 – *How to Use Layers* in the *Design Guide*. Note that this layer condition only refers to the individual Grammar item. A precondition for the entire Grammar object can be defined within the top-level **Precondition** section of the **Grammar** editor.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Language	Defines the language layer the grammar belongs to. At call time, the server only activates the grammars of the currently active language layer, e.g. if the currently active language is *en-UK* then only grammars on either the language layer *English (United Kingdom)* or *English* are activated.
Channel	Defines the channel layer the grammar belongs to. At call time, the server only activates the grammars of the currently active channel layer., e.g. if the currently active channel is *voice* then only grammars with the channel layer set to *voice* will be processed. If a grammar is valid for the voice and video channel, or for the text and Web channel, use the combined entries *Voice/Video* or *Text/Web*, respectively.
Weighting	Optional parameter to define the relative weighting of this grammar compared to other grammars active at a certain dialog step. This can be used to indicate that some caller inputs are more or less likely to occur than others. The actual effect depends on the way the underlying media platform has implemented this feature. This setting is only relevant in the voice and video channel. If the weight is specified via a Variable, Expression, Layer, or Script object, the weight factor needs to be specified as a numerical value, e.g. 0.2 for 20% or 5.0 for 500%. Note that this field is a combo field, i.e. you can also type in any arbitrary real number.

Settings in these fields apply to all subsections specified later on (see below).

If multiple Grammar items exist for the same language and layer in the voice or video channel, all of them are rendered into the resulting VoiceXML. The media platform then merges them into one combined grammar. In the text and Web channel, only the first definition (in object editor order) will be used.

In addition, each **Grammar Item** section, contains the following three subsections (represented by tabs in Desktop for Web):

- Voice/Text
  To define a voice grammar in the voice and video channel, or to define a text grammar in the text and Web channel.
- DTMF
  To define a DTMF grammar in the voice and video channel. In the text and Web channel, any definition in this section will be ignored.
- Preprocessing
  To define initial processing that dynamically builds a grammar at call time.

Voice/Text

In the **Voice/Text** section, the content of the grammar is defined. In case of the voice and video channel you define a grammar for voice input, which is specified through an embedded definition that is stored within the Grammar object, or through a reference to an external file. In case of the text and Web channel, you define a grammar for text input, which needs to be an embedded definition.

Copyright © 2001-2011 Voxeo Germany GmbH

Embedded Grammar Definition
For an embedded definition, you may type in the grammar code into the **Grammar** field (or text area in Desktop for Web). By default, the **TTG** check box is selected, indicating that the grammar is defined in text-to-grammar format. This enables you to define grammars via comma-separated lists of utterances without having to bother about specific grammar formats appropriate for the underlying media platform. TTG is the only format supported when writing grammars in the text and Web channel. More details can be found in the Text-to-Grammar paragraph.

If you clear the **TTG** check box, you need to make sure that the grammar format you provide is understood by the media platform you are deploying your application on. For more details, see the Grammar Formats paragraph.

In the voice and video channel, you can explicitly use a `<grammar> … </grammar>` tag pair in your embedded non-TTG grammar definition; if you do so, the definition is copied verbatim into the VoiceXML code that is generated by the server at call time. Note, however, that if XML-related characters such as "`<`" `or` "`>`" are used within the grammar code itself, it needs to be wrapped in CDATA.
If your embedded definition does not use a `<grammar>` tag, the server automatically wraps it at call time, inserting the appropriate settings for grammar type, grammar weighting, etc.

Embedded grammar definitions can also be provided through references to Variable, Expression, Collection, Layer, or Script objects. If an object reference is used as an embedded definition, the respective object is evaluated at call time and its value in turn becomes the definition for the grammar.

External Grammar Reference
Besides using embedded definitions, the voice and video channel allows a grammar to be defined by referencing an external file. To do so, specify the following:

Property	Description
Location	Optional parameter to specify a location (using a Resource Locator object).
File	Defines the filename for the grammar file to be used.
Rule	Optional parameter to define the starting rule to be used within the grammar file.
Extension	Defines the file extension for the grammar, which is appended to the filename. If you include the extension in the **File** definition, set the extension to *None*. If you define an extension other than *None*, this setting can be overwritten by the corresponding setting in the service definition. For more information on the definition of services, refer to Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*.



Caution: Not all media platforms support the use of grammar files with multiple public rules that can be accessed using the **Rule** field. Some media platforms only allow one public rule per grammar file. In this case, the **Rule** field should be left empty.

The URL that is created within the markup code to reference the grammar at call time has the following structure:

```
[Location] + [File] + [#] + [Rule]
```

The "#" character is only inserted if both a file and a rule are defined.

Copyright © 2001-2011 Voxeo Germany GmbH

In a similar manner as described for the embedded grammar definition, filename and rule name can also be defined through a Variable, Expression, Layer, or Script object.

Tip: When working with Voxeo Prophecy and a third party ASR engine, external grammar files are usually loaded by Prophecy and converted to a format appropriate for the ASR engine. This may prevent the ASR engine from caching the file reference. To bypass the conversion mechanism, use the Tuning property **Resource Fetching – Grammar Fetch Options** to set *voxeo:useuri=true*. This forces Prophecy to simply send the file reference itself to the ASR engine. Note that in this case the grammar file must be in a format that can be understood directly by the ASR engine.

Precedence
In case both an external grammar reference and an embedded grammar definition are present, use the **Precedence** setting to decide which one to use. Select *Embedded:External*, if the embedded definition is to be used. Select *External:Embedded*, if the external reference is to be used. If you leave the setting at *Default*, the Service object definition of this property is used. See Configuring a Service in Chapter 2 – *Configuring Servers and Services* of the *Deployment Guide* for more information on how to set the grammar mode in a Service object.

Note: If you want to add more rules to an existing external grammar, you can define these extra rules as an embedded grammar and use the marker @SOURCE@, which will be replaced with the resource locator path, the filename, and the extension of the external file definition at call time.

Type
The **Type** field defines the format of the grammar and is only relevant in the voice and video channel. A corresponding attribute is rendered into the VoiceXML `<grammar>` tag at call time. A **Type** field set to *Default* indicates that the grammar format defined in the corresponding Service object is used. For more details, refer to Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*. If no format is to be set, select *None*. Note that only those types are allowed that are supported by the media platform. If you select a type that is not supported, the server renders the default grammar type for the selected platform instead.

Built-in grammars
Special types of grammars are *built-in* for standard inputs such as yes/no, a date, a phone number etc. The support for built-in grammars depends on the media platform used; in general, only voice and video applications based on VoiceXML can support built-in grammars. Check the documentation of your media platform to see which built-in grammars are provided. See Appendix A – *Media Platform Drivers* in the *Deployment Guide* to find out if your platform supports built-in grammars at all. Note that *built-in* is not the same type as *precompiled*. *Precompiled* grammars are a special type of grammar offered by some media platforms. To make use of built-in grammars in Input objects, set the **Type** field to *Built-in* and leave the **Slot name** field in the **Result Handling** empty. In the text area for embedded grammars, enter the type of grammar required. The following types are possible: *boolean*, *date*, *digits*, *currency*, *number*, *phone*, and *time*. Proprietary types provided by the platform can also be used. For more information on the standard types, refer to the VoiceXML 2.0 specification at http://www.w3.org/TR/voicexml20, Appendix P – *Builtin Grammar Types*.
For voice and DTMF grammars in the voice and video channel, the type *digits* can be parameterized in order to set a minimum and/or maximum length for digit sequences. Use *minlength*, *maxlength*, and *length*, to set the minimum, maximum, or the exact length, respectively.
For DTMF grammars, the type *boolean* can be parameterized in order to define which key corresponds to an affirmative and a negative answer. Use *y* and *n* to set the keys for *yes* and *no*, respectively.
The parameters are appended to the type itself.

Copyright © 2001-2011 Voxeo Germany GmbH

Examples:

digits?length=4

digits?minlength=2;maxlength=4

boolean?y=1;n=0

DTMF

In the **DTMF** section, a DTMF grammar can be defined for touchtone input using the keypad of the phone. Any definition made here will only be applied in the voice and video channel.

The corresponding fields are structurally the same as for Voice/Text, and the same mechanisms apply. Note, however, that according to the VoiceXML specification the weight entry (defined in the **Weighting** field) has no effect for DTMF grammars.

Note: When using the hash key '#' in a TTG grammar, you need to set the tuning parameter *DTMF – Termination Character* to "" (the empty string). Otherwise the media platform will interpret the hash key '#' as the termination key to end a sequence of DTMF keys, and not as a key on its own.

Some media platforms allow defining DTMF input within a voice grammar. For these platforms, the use of the **DTMF** section is optional. For portability reasons, it is recommended to use it, though. And in case your application only allows one input mode at a time (either voice *or* DTMF), VoiceObjects Server renders only the grammar that corresponds to the currently active input mode.

Caution: Some media platforms do not support slot assignments in DTMF grammars. These platforms always return the DTMF keys pressed, but no semantic interpretation. If you use TTG for DTMF grammars, this will be handled by the server, i.e. in this case slot assignments will work as usual. If you use external, or inline but non-TTG grammars, you need to be aware of this restriction. See Appendix A – *Media Platform Drivers* in the *Deployment Guide* to find out how your media platform handles this.

Preprocessing

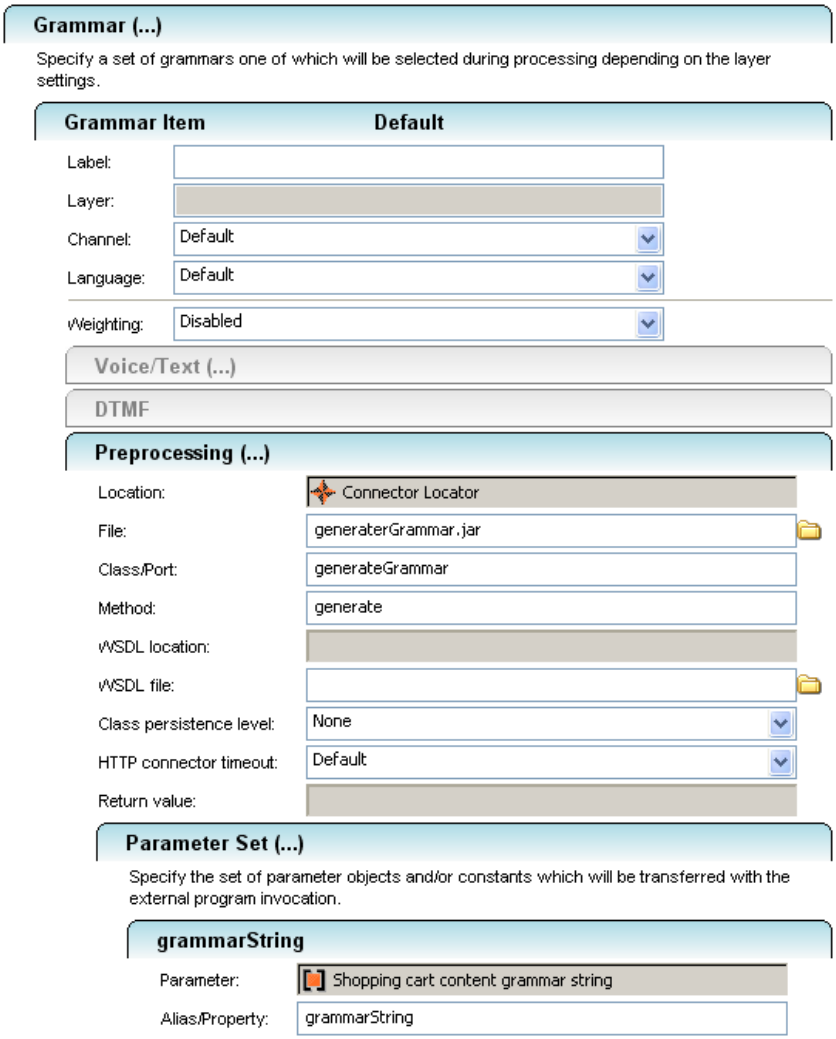
The **Preprocessing** section enables the dynamic generation of grammars at call time and is supported in all four channels. In certain cases, e.g. when maintaining a shopping cart within an application that the user can remove items from, the specific entries in a grammar are not known at design time.

Both the **Voice/Text** and the **DTMF** definition can make use of Variable, Expression, Collection, Layer, or Script objects. At call time, the server first processes whatever is defined in **Preprocessing**, and then evaluates the **Voice/Text** and **DTMF** sections to generate the appropriate grammar definitions. This enables you to fill variables with certain values within the **Preprocessing** and then use those values in the **Voice/Text** or **DTMF** definitions.

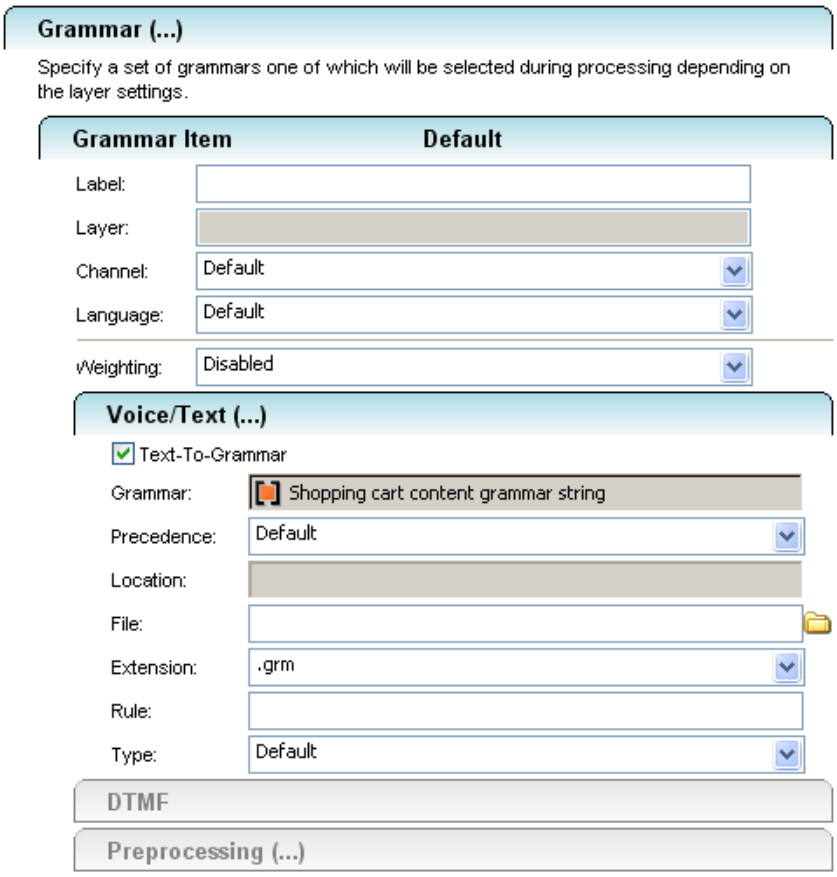
For a definition of the individual fields in the **Preprocessing** section, refer to the Connector object.

The figures below provide an example for the handling of a shopping cart. The **Preprocessing** section specifies a Java connector that fills a Variable object called *Shopping cart grammar* with the appropriate grammar. This Variable object is then used as an embedded definition in the **Voice/Text** section.

Copyright © 2001-2011 Voxeo Germany GmbH



Copyright © 2001-2011 Voxeo Germany GmbH



Text-to-Grammar

When building voice or video applications in a rapid prototyping approach, it is often desirable to start the definition of a grammar by just providing a couple of typical utterances a caller might say. The application designer does not need to know the exact grammar syntax required by the media platform. The text-to-grammar (TTG) functionality is built to support this approach. A TTG definition consists of a comma-separated list of possible caller input, potentially enriched with slot values.

While in the voice and video channel TTG is one of several options to define grammars, in the text and Web channel it is the only supported format.

The following paragraphs apply to all channels and describe how to build TTG grammars. Find examples for the various channels at the end of this paragraph.

The tokens of a TTG grammar may contain multiple words, as in the following example:

car insurance, life insurance, health insurance

In certain cases it is desirable to define a slot return value that is associated with a specific utterance. This can be done by providing the slot value within parentheses behind each token:

car insurance (car), life insurance (life), health insurance (health)

If no slot values are defined, the utterance itself is returned as the slot value.

Copyright © 2001-2011 Voxeo Germany GmbH

When using TTG, the result is returned in a default slot called *sltTTG* for all media platforms certified with the VoiceObjects platform (note, though, that it is possible to define a different slot name through a media platform driver setting if required). This slot name must be used in the **Result Handling** of an Input object to assign it to a Variable or Layer object. It is also possible to define different slot names within one TTG grammar, using a '#' notation within the parentheses:

car insurance (#sltChoice# car), life insurance (#sltChoice# life), health insurance (#sltChoice# health)

In this example, all three utterances fill a slot *sltChoice*. For reasons of convenience, you can define the slot name once and then leave it out for all following utterances:

car insurance (#sltChoice# car), life insurance (life), health insurance (health), premium customer (#sltCustomer# premium), gold customer (premium)

In this example, saying *car insurance*, *life insurance* or *health insurance* will fill the slot *sltChoice* with *car*, *life*, or *health*, respectively, and saying *premium customer* or *gold customer* will fill *sltCustomer* with *premium*.

Some ASR engines support *garbage matching*; this enables the media platform to match any kind of caller input between certain keywords. In TTG, you can make use of garbage by working with the placeholder @GARBAGE@. This placeholder can be put at any position within one utterance. Consider the following example:

@GARBAGE@ car insurance, @GARBAGE@ life insurance, @GARBAGE@ health insurance

This grammar matches utterances such as "*I need information on car insurance*", "*Please give me info on life insurance*", "*life insurance*", etc. If your media platform does not support garbage, the placeholder is removed and has no further effect. Consult your media platform vendor for more information on this.

In the text and Web channel, the utterances are displayed on the screen as options, in case the tuning parameter *Presentation – Input* is set to *radio*, *list*, or *menu*. To be able to use special characters like commas or parentheses in the output, which are reserved characters from the TTG syntax, you can use the back-slash symbol for escaping. Consider the following example, where the grammar is to be displayed as a *menu*:

Monday\, 3rd Jan \(today\) (today), Tuesday\, 4th Jan \(tomorrow\) (tomorrow)

This would be shown as "*Monday, 3rd Jan (today)*" and "*Tuesday, 4th Jan (tomorrow)*", with the slot values *today* and *tomorrow*, respectively.

If the tuning parameter *Presentation – Input* is set to *radio* or *list* in a Web channel application, it is desirable to mark one utterance as pre-selected, so the browser can display it accordingly. This can be achieved by using the marker @SELECT@ within the parentheses, in front of potential slot assignments. Have a look at the following examples:

male (male), female (@SELECT@female)

male (#Gender#male), female (@SELECT@#Gender#female)

These are functionally identical, with the difference that the second one mentions the slot name explicitly. By using the @SELECT@ marker for *female*, the browser will be instructed to pre-select this option in the display.

In the Web channel, multi-slot Input objects are possible. The TTG grammar must define the set of options per slot in this case. The following example shows how to do this:

male(#Gender#), female (@SELECT@), ? (#Age#)

Copyright © 2001-2011 Voxeo Germany GmbH

This TTG grammar requires two slots in the corresponding Input object named *Gender* and *Age*. If these are not present, processing of this object will fail. For the slot called *Gender*, two options are defined. If no value for the tuning parameter *Presentation – Input* is defined for this slot name, the server will show these options as radio buttons. For the *Age* slot, the server will generate an input field for free entry of text. At the bottom of the resulting page, a submit button is shown so that the caller can apply their selection and submit that.

TTG can also build grammars on the basis of Collection objects. For single-slot grammars using the standard slot *sltTTG*, this can be done in two different ways: If the collection has a single column (named *utterance*), then this column is used for both utterances and slot values as described in the first case above. The corresponding collection would look like this:

```
<root>
	<row><col name=″utterance″>car insurance</col></row>
	<row><col name=″utterance″>life insurance</col></row>
	<row><col name=″utterance″>health insurance</col></row>
</root>
```

Alternatively, slot values and multi-lingual grammars can be defined via multi-column collections like this:

```
<root>
	<row>
		<col name=″en-US″>car insurance</col>
		<col name=″de-DE″>autoversicherung</col>
		<col name=″key″>car</col>
	</row>
	<row>
		<col name=″en-US″>life insurance</col>
		<col name=″de-DE″>lebensversicherung</col>
		<col name=″key″>life</col>
	</row>
	<row>
		<col name=″en-US″>health insurance</col>
		<col name=″de-DE″>krankenversicherung</col>
		<col name=″key″>health</col>
	</row>
</root>
```

In this format, the column “key” defines the slot return value while columns with language code names such as “en-US” or “de-DE” define the corresponding utterances. A column “default” may be provided, which is used if no matching column exists for the active language setting. If more than one utterance should return the same slot value, separate rows with the same “key” value need to be defined, i.e. you cannot use a comma-separated list of utterances in an utterance column.

For grammars that need to set more than one slot, or single-slot grammars that use a slot name other than *sltTTG*, you can add columns “slotname” that define, for every row, the name of the slot to be filled. An example is:

Copyright © 2001-2011 Voxeo Germany GmbH

```
<root>
	<row>
		<col name="en-US">car insurance</col>
		<col name="de-DE">autoversicherung</col>
		<col name="slotname">sltChoice</col>
		<col name="key">car</col>
	</row>
	<row>
		<col name="en-US">life insurance</col>
		<col name="de-DE">lebensversicherung</col>
		<col name="slotname">sltChoice</col>
		<col name="key">life</col>
	</row>
	<row>
		<col name="en-US">health insurance</col>
		<col name="de-DE">krankenversicherung</col>
		<col name="slotname">sltChoice</col>
		<col name="key">health</col>
	</row>
</root>
```

Note: It is not possible to simply enter the code of the collection into the **Grammar** field within the **Grammar** editor. If you want to use it for TTG you need to insert the actual Collection object.

Using TTG in the voice and video channel has the important advantage that grammars are automatically created in a format appropriate for the media platform used. In particular, media platforms can be switched without having to make any changes to the application itself.

The grammar type in which the TTG grammar should be created (e.g. *Nuance GSL* or *SRGS XML*) can be determined in a variety of ways. The recommended approach is to define the desired type in the corresponding Service object, and leave all individual Grammar items set to *Default*.

Not all media platforms support all of the available grammar types. Therefore, if the **TTG** check box is selected and a grammar type is chosen that is not known to be supported by the current media platform, the server reverts to the default grammar type for the media platform. This is to ensure that TTG definitions always produce grammars that work with the underlying media platform.

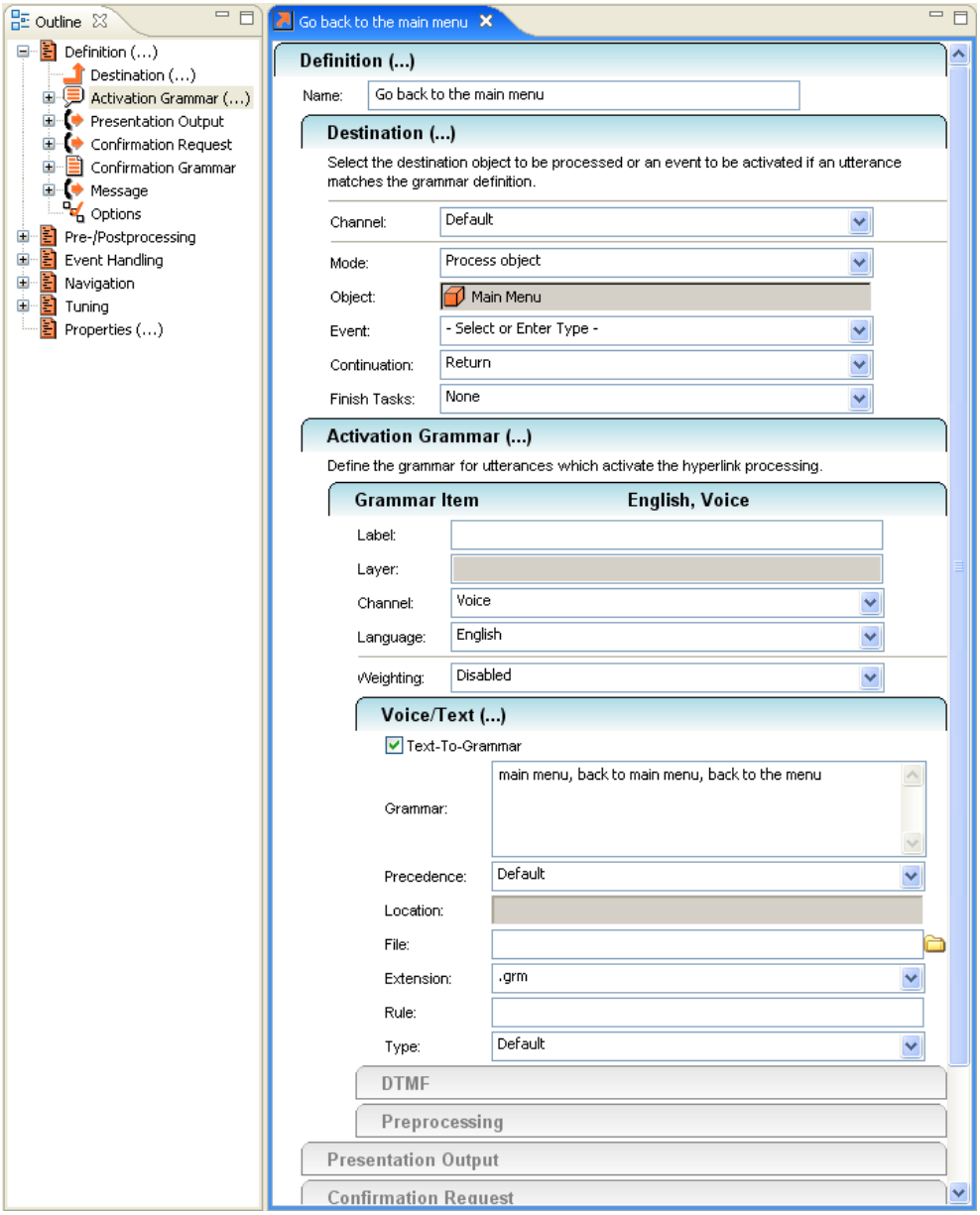
Finally, it should be kept in mind that TTG is intended for rapid prototyping and small to medium-sized grammars. As TTG in the voice and video channel creates an inline `<grammar>` element embedded within the VoiceXML code sent to the media platform, no caching of these grammars is possible. Therefore, large grammars should reference external files, which will enable caching and improve overall system performance.

Note: When using the hash key '#' in a TTG grammar, you need to set the tuning parameter *DTMF – Termination Character* to "" (the empty string). Otherwise the media platform will interpret the hash key '#' as the termination key to end a sequence of DTMF keys, and not as a key on its own.

Copyright © 2001-2011 Voxeo Germany GmbH

The following examples illustrate different uses of TTG, depending on the channel.

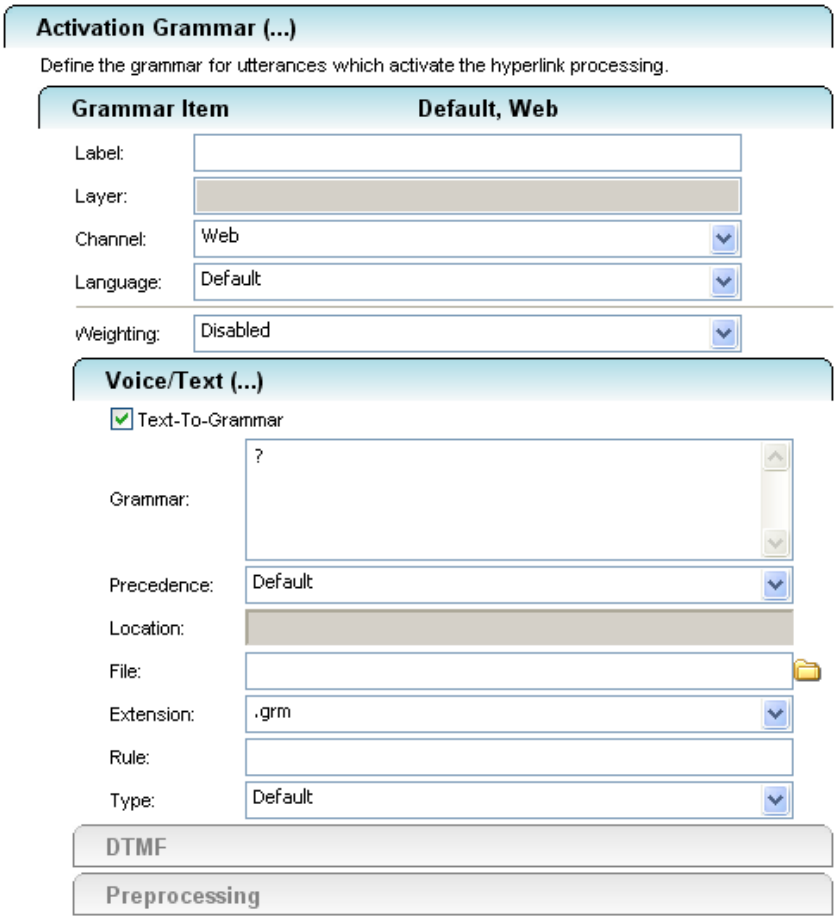
The first example shows the use of TTG in a Grammar object embedded in a Hyperlink object that allows jumping back to a top-level menu. Here, no slot is required. The TTG grammar is defined for the voice channel.



The same Hyperlink object could be re-used for the text and Web channel, by defining appropriate additional Grammar items using TTG.
In the text channel, the grammar will be shown as the access key of the option that will be generated for the hyperlink by the USSD browser. See Chapter 10 – *How to Support Multiple Phone Channels* in the *Design Guide* for examples of how the display of a text application could look like. Therefore, define some digit as the grammar for this hyperlink, e.g. *0* (zero).
In the Web channel, the grammar will not be processed at all, only the presentation output will be used to display the hyperlink. Nevertheless, a grammar definition is mandatory in all channels. So either re-use the text channel grammar definition in

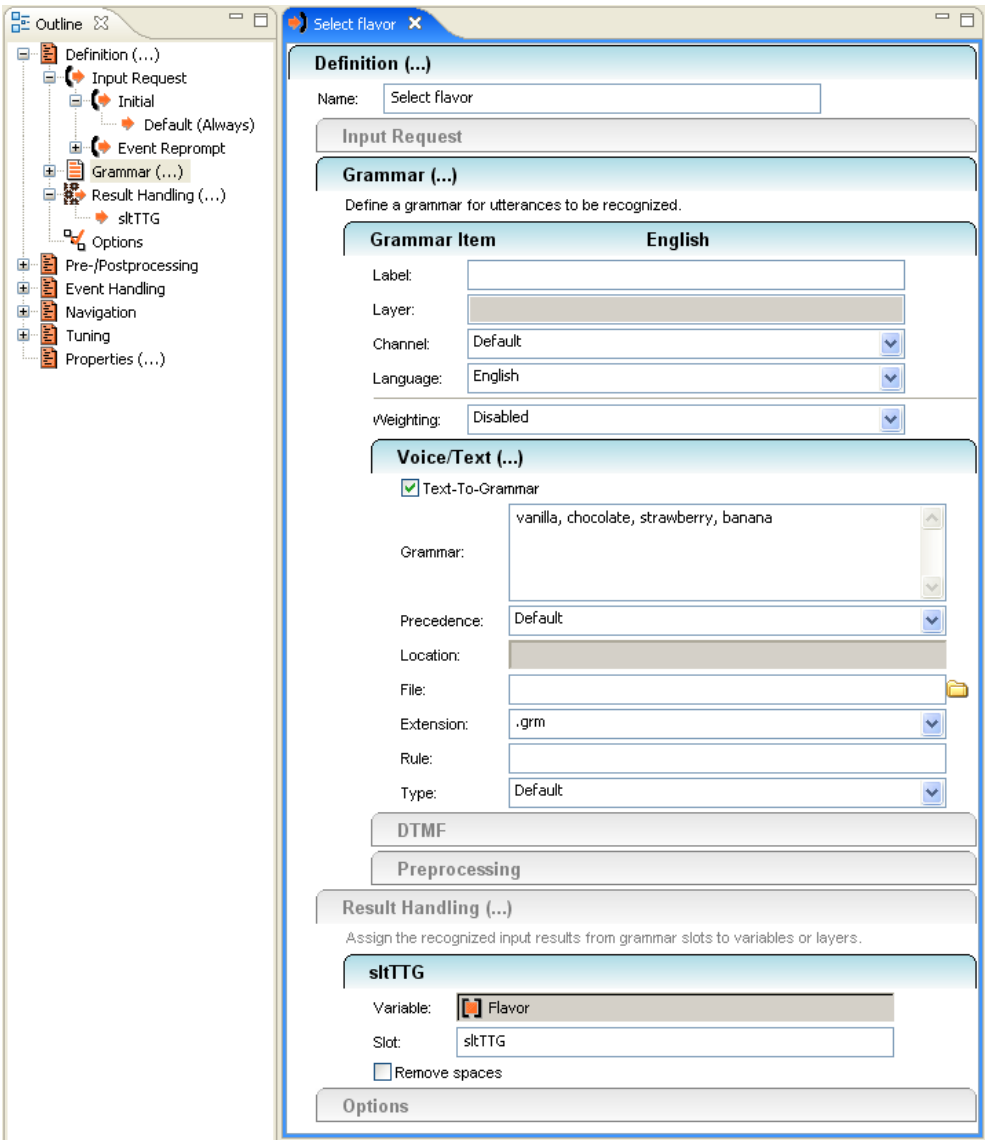
Copyright © 2001-2011 Voxeo Germany GmbH

case your application should support both channels, or use the question mark “?” as a placeholder definition for a pure Web grammar:



The second example shows an Input object that makes use of the slot *sltTTG* without explicitly defining slot values in the TTG grammar definition. After the caller has selected a flavor, the utterance itself is returned in the slot. Thus the Variable object *Flavor* contains *vanilla*, *chocolate*, *strawberry*, or *banana*.

Copyright © 2001-2011 Voxeo Germany GmbH

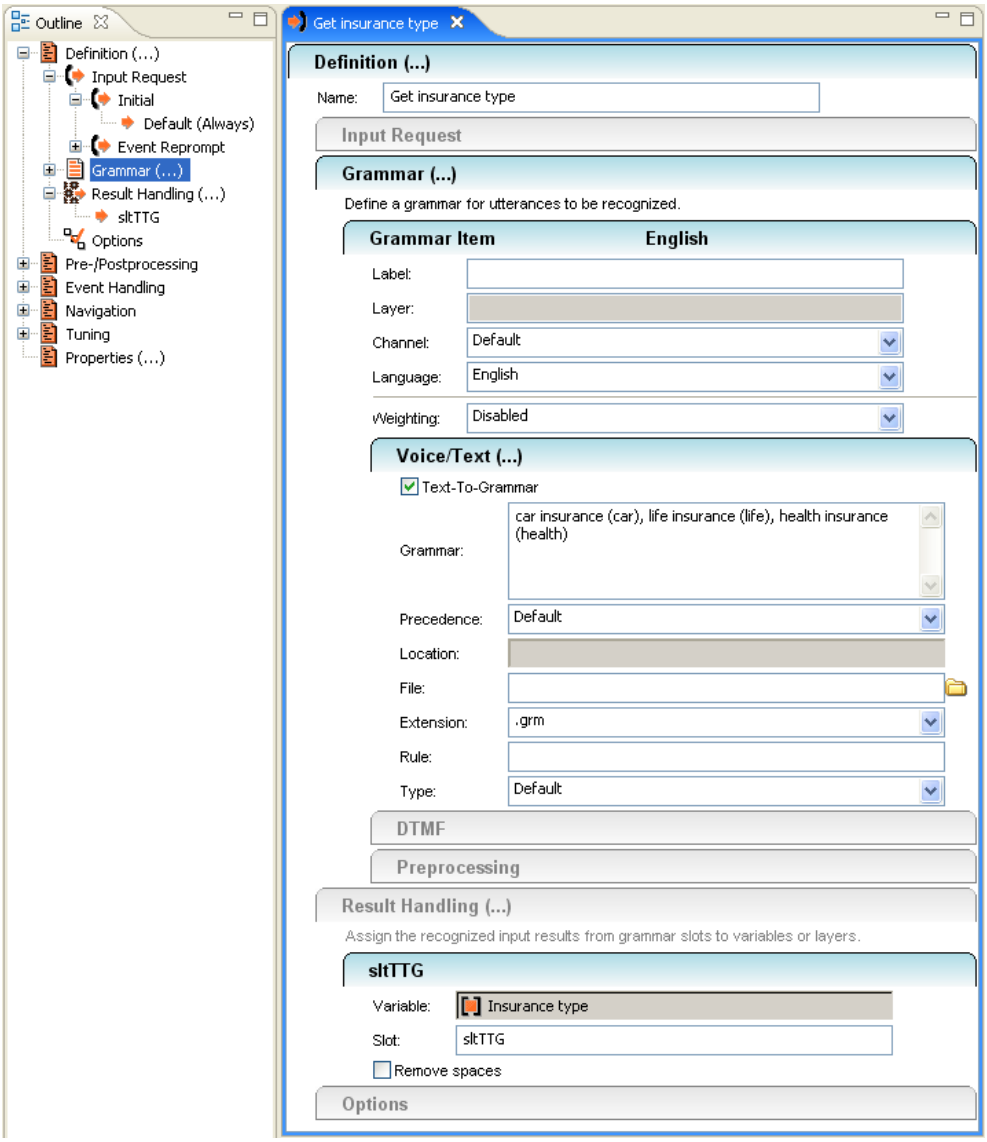


Note that in case of a single slot name, the slot name definition can be left empty, for convenience reasons. Since TTG grammars provide one slot only, you can always leave the **Slot name** field empty when making use of TTG grammars.

This grammar works in all four channels, therefore the **Channel** setting is left at *Default*. In the voice and video channel, it will allow the caller to say one of the four mentioned flavors. In the text and Web channel, a TTG grammar defining between 1 and 5 utterances will be displayed on the screen as a set of options, as it is best practice to rather offer the caller a range of predefined choices than to let them type in their response manually. How the grammar will be shown (e.g. as hyperlinks, radio-buttons or a drop-down list in the Web channel) can be defined using the tuning parameter *Presentation – Input* and is described in more detail in the Input object in this *Object Reference*.

The final example shows an Input object that makes use of explicit slot values, which is again supported by all channels. After the caller has chosen an insurance type by saying or selecting *car insurance*, *health insurance*, or *life insurance* the Variable object *Insurance type* contains the slot value *car*, *health*, or *life*.

Copyright © 2001-2011 Voxeo Germany GmbH



Note: Not all voice and video platforms can handle blanks in slot return values. Thus when using utterances in TTG that contain blanks, it may be required for these platforms to provide an explicit slot return value for this utterance that does not contain blanks, e.g. “car insurance (car_insurance)”.

Utterance Validation

When building applications for the text or Web channel, there sometimes is the situation that the user is asked to provide free-form input that still needs to meet certain criteria, such as e.g. an account number that must be a seven digit number not starting with 0.

While the application can certainly perform such validation and loop back to the respective input state if it is not met, it is often more desirable to have a concept similar to the No Match condition known from the voice channel. This can be achieved by using the utterance validation mechanism built into VoiceObjects Server.

To enable utterance validation, the grammar definition must be provided in the format

Copyright © 2001-2011 Voxeo Germany GmbH

```
regex:RegularExpression:regex (#slotName#)
```

where `RegularExpression` represents any regular expression and `slotName` stands for the name of the slot to be validated. Multiple such statements may be used, separated by comma, for multi-slot inputs in the Web channel (though only a single regular expression is allowed for each individual slot). For single nameless slots, the `slotName` part may be omitted.

As an example, if a single slot input is supposed to be a seven digit number not starting with 0 then the following grammar definition could be used:

```
regex:[1-9][0-9]{6}:regex
```

If the caller's input matches the regular expression, the dialog proceeds normally with the next object in the dialog flow.

If the input does not match the regular expression, a No Match event is triggered. This can be handled in the usual way using an event handler; for more information refer to Event Handling. Note that no event counter is kept, so multiple iterations of invalid caller inputs will always trigger the same No Match event handler. If different behavior is desired for different levels, the application may keep its own counter within the No Match event handling.

In the No Match handler, you can use the expression LASTRESULT(*validation*) to retrieve a comma-separated list of those slots where input validation against the respective regular expressions failed. LASTRESULT(*validation, Slotname*) will return *true* if validation succeeded for the specified slot, *false* if it failed.

Grammar Formats

Three major grammar formats are currently in use in voice and video applications: GSL, JSGF, and XML. The table below provides a sample for each of them.

Grammar Format	Sample Grammar
GSL	Highway (?[freeway highway route] [(one oh one) {<sltHighway "101">} (one hundred one) {<sltHighway "101">} (one hundred and one) {<sltHighway "101">} (two eighty) {<sltHighway "280">} (two hundred eighty) {<sltHighway "280">} (two hundred and eighty) {<sltHighway "280">} (six eighty) {<sltHighway "680">} (six hundred eighty) {<sltHighway "680">} (six hundred and eighty) {<sltHighway "680">}])

Copyright © 2001-2011 Voxeo Germany GmbH

Grammar Format	Sample Grammar
JSGF	public <Highway> = [freeway | highway | route] [(one oh one) {this.sltHighway="101"} | (one hundred one) {this.sltHighway="101"} | (one hundred and one) {this.sltHighway="101"} | (two eighty) {this.sltHighway="280"} | (two hundred eighty) {this.sltHighway="280"} | (two hundred and eighty) {this.sltHighway="280"} | (six eighty) {this.sltHighway="680"} | (six hundred eighty) {this.sltHighway="680"} | (six hundred and eighty) {this.sltHighway="680"} |];
SRGS XML	<grammar xml:lang="en-US" version="1.0" root="Highway"> <rule id="Highway" scope="public"> <item repeat="0-1"> <one-of> <item> freeway </item> <item> highway </item> <item> route </item> </one-of> </item> <one-of> <item tag="sltHighway='101'"> one oh one </item> <item tag="sltHighway='101'"> one hundred one </item> <item tag="sltHighway='101'"> one hundred and one </item> <item tag="sltHighway='280'"> two eighty </item> <item tag="sltHighway='280'"> two hundred eighty </item> <item tag="sltHighway='280'"> two hundred and eighty </item> <item tag="sltHighway='680'"> six eighty </item> <item tag="sltHighway='680'"> six hundred eighty </item> <item tag="sltHighway='680'"> six hundred and eighty </item> </one-of> </rule> </grammar>

The example grammars shown above each fill a slot called *sltHighway* with the number of the highway the caller entered. Note that while the utterances themselves are defined textually (*six eighty*), the slot values are digit strings (*680*). This is preferable in order to abstract the semantic interpretation from the actual utterances, and in order to simplify the business logic used within the application itself. Using the

Copyright © 2001-2011 Voxeo Germany GmbH

numbers instead of the text makes it very easy, for instance, to run exactly the same application in Spanish with only the utterances defined in the grammar being different.

Note: Some media platforms require the explicit tagging of grammars with a language code. This implies that when using an embedded grammar definition containing explicit `<grammar> … </grammar>` tags, you are responsible for providing this tag attribute. As a convenience, you may use the marker @LANGUAGE@ which the server will replace with the currently active language at call time when creating the markup code.

VoiceObjectsXML Definition

The Grammar object is represented by the VoiceObjectsXML element `<grammar>`. It has two groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<grammar>` element uses the embedded `<grammarItem>` element.

Grammar

Children

- **`<expression usage="precondition">`** or **`<variable usage="precondition">`** or **`<collection usage="precondition">`** or **`<script usage="precondition">`**
  Defines the precondition for the Grammar object.
- **`+<grammarItem>`**
  Defines the list of Grammar items.

Example

```
<grammar>
  <grammarItem language="en-US">
    <grammarDefinition mode="voice">
      one, two, three
    </grammarDefinition>
    <grammarDefinition mode="dtmf">
      1, 2, 3
    </grammarDefinition>
  </grammarItem>
  <grammarItem language="de-DE">
    <grammarDefinition mode="voice">
      eins, zwei, drei
    </grammarDefinition>
    <grammarDefinition mode="dtmf">
      1, 2, 3
    </grammarDefinition>
  </grammarItem>
</grammar>
```

Copyright © 2001-2011 Voxeo Germany GmbH

GrammarItem

Attributes

- **`label`**
  A text string providing a name for the Grammar item.
- **`language`**
  Defines the language for the Grammar item. Can be *default* or a valid language code (e.g. *de-DE*, *en-US*, etc.). If not specified, defaults to *default*.
  Appendix A – Language Codes contains a list of all language codes available in VoiceObjects together with the respective language they represent.
- **`layer`**
  Defines the layer for the Grammar item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **`channel`**
  Defines the channel(s) for which this Grammar item is valid. Can be *default*, *voice*, *video*, *text*, *web*, *voiceVideo*, or *textWeb*. If not specified, defaults to *default*.
- **`weighting`**
  Defines the relative weighting of the grammar. Indicated by *disabled* or a positive float value. E.g. 0.5 indicates 50% weight, and 1.7 indicates 170% weight. If not specified, defaults to *disabled*.

Children

- **`<grammarDefinition mode=”voice”>`**
  Defines the Grammar item section for voice.
- **`<grammarDefinition mode=”dtmf”>`**
  Defines the Grammar item section for DTMF.
- **`<connectorItem>`**
  Defines the preprocessing for the Grammar item. For more information, refer to the Connector object.

Example

```
<grammarItem>
  <grammarDefinition mode=”voice” ttg=”false” grammarType="xml">
    <![CDATA[<rule id="root" scope="public">operator</rule>]]>
  </grammarDefinition>
</grammarItem>
```

GrammarDefinition

Attributes

- **`mode`** [required]
  Indicates the mode of the Grammar object. Can be either *dtmf* or *voice*.
- **`location`**
  Defines the location from where the grammar is to be retrieved. Must be a reference to a Resource Locator object.

Copyright © 2001-2011 Voxeo Germany GmbH

- **`file`**
  Defines the grammar file name, without extension. Can be a constant name, or a reference to an Expression, Script, or Variable object.
- **`rule`**
  Defines the rule name to be used within the grammar file. Can be a static name, or a reference to an Expression, Script, or Variable object.
- **`grammarPrecedence`**
  Indicates if the embedded or external grammar definition takes precedence, in case both are present. Must be one of *embedded:external* or *external:embedded*, or *default*. If not specified, defaults to *default*.
- **`grammarType`**
  Defines the type of the grammar. Must be one of *default*, *abnf*, *gsl*, *jsgf*, *xml*, *cisco*, *precompiled*, *builtin*, *none*. If not specified, defaults to *default*.
- **`embeddedRef`**
  Defines an embedded grammar through a reference to an Expression, Script or Variable object.
- **`ttg`**
  Indicates whether to use TTG on the embedded definition. Ignored if no embedded definition is present. May be *true* or *false*. If not specified, defaults to *true*.
- **`grammarFileExtension`**
  Specifies the file extension for the grammar file as either a constant value or a reference to an Expression, Script or Variable object. Legal constant values are *none*, *fsg*, *gram, grammar*, *grm*, *grxml, gsl, jsgf*, *ngo*, *sjv*, *srgs*, *txt*, *xml*. If not specified, defaults to *grm*.

Children

- **CDATA**
  Optional constant embedded grammar definition.

Examples

```
<grammarDefinition mode=”voice” ttg=”true”>
  apple, orange, pear
</grammarDefinition>


<grammarDefinition mode=”voice” embeddedRef="#MainMenu" >


<grammarDefinition location=”#Grammar Locator” file=”cars”
grammarFileExtension=”gsl” grammarPrecedence=”external:embedded”>
  bmw, ford, nissan
</grammarDefinition>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The following table contains all object types that can reference a Grammar object:

Icon	Object Name	Use Case Example
	Input	A Grammar object can be used within an Input object.
	Menu	A Grammar object can be used within a Menu item of a Menu object.
	Confirmation	A Grammar object can be used within a Correction item of a Confirmation object.
	List	A Grammar object can be used for the navigation and selection grammars in the List object.
	Hyperlink	A Grammar object can be used within a Hyperlink object.
	Pause	A Grammar object can be used as the wake up grammar within a Pause object.
	Transfer	A Grammar object can be used as the termination grammar within a Transfer object.
	OSDM	A Grammar object can be linked in the parameter set within any OSDM object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Grammar object should indicate which types of utterances the grammar covers. The short description should contain a brief explanation of the possible caller utterances, as well as assumptions or restrictions (e.g. the language covered). The table below lists three examples:

Name	Description
Yes/No grammar	Allows the caller to flexibly answer yes/no questions in a variety of ways. US-English.
Date grammar	Allows the caller to enter dates in both absolute format (*June first two thousand four*) and relative format (*next Monday*). German.
Mobile phone number grammar	Allows the caller to enter German mobile phone numbers with the area codes 0170, 0171, 0172, and 0173. German.

Copyright © 2001-2011 Voxeo Germany GmbH

Hyperlink

Overview

The Hyperlink object allows callers to branch out to a given dialog component at any point in time through predefined commands, which activate the link. In addition, it also provides the capability for integrated confirmation in order to prevent the unwanted activation of a hyperlink, e.g. due to misrecognition.
Conceptually, a hyperlink in voice or video applications can be compared to a hyperlink on a Web page. The Hyperlink object is supported in all four channels.

The Object Definition below covers the configuration of the Hyperlink object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML, refer to the VoiceObjectsXML Definition paragraph.

The Hyperlink object belongs to the object category Actions.

Dialog Flow Scenario

The following dialog flow example presents the use of a Hyperlink object within the *Voice Shopping* portal. It branches out to the *Entertainment* portal and is activated by the caller through a particular command.

Object – Caller	Dialog Flow
	*Hello and welcome to the Voice Shopping portal.*
	*Switch to the Entertainment portal.*
	*Hello and welcome to the Entertainment portal. Say games to go to the game section, jokes to hear some funny jokes or music to go to the jukebox.*

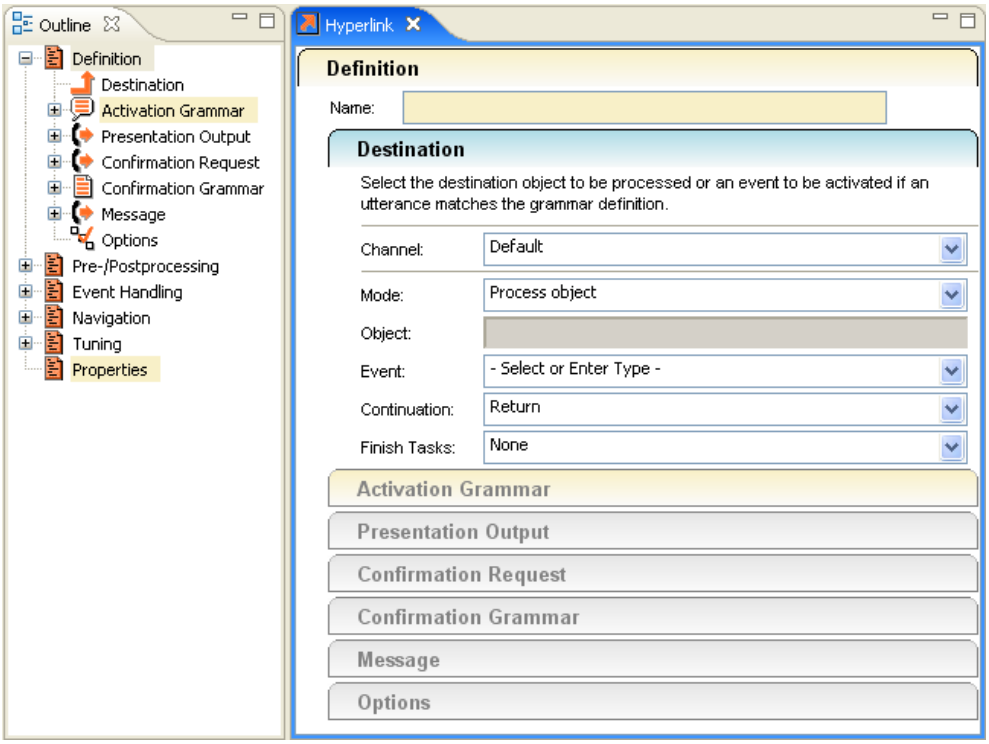
Object Definition

The **Definition** of the **Hyperlink** object provides the following sections:

- Destination
  To provide the behavior upon activation of the hyperlink and specify how to proceed with the call after the link has been processed.
- Activation Grammar
  To define the grammar for the input that activates the Hyperlink object.
- Presentation Output
  To define an output that presents this hyperlink to the caller. This output is mandatory in the text and Web channel and optional in the voice and video channel.
- Confirmation Request
  To optionally request a confirmation from the caller before activating the Hyperlink object.
- Confirmation Grammar
  To define the grammars for denying or confirming the activation of the Hyperlink object.

Copyright © 2001-2011 Voxeo Germany GmbH

- Message
  To define optional outputs for the following cases:
  - **Activation**: to present an output before the hyperlink destination object is processed.
  - **Deactivation**: to present an output if the caller explicitly denies processing of the hyperlink destination object.
  - **Return**: to present an output before returning to the dialog flow after the hyperlink destination object has been processed.
- Options
  To define various processing options.



For information regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Navigation*, *Tuning*, and *Properties* in this *Object Reference*.

Objects defined within preprocessing are processed immediately after activation of the hyperlink by the caller, i.e. prior to anything defined in the Hyperlink object itself. The postprocessing is done after the entire Hyperlink object has been processed, i.e. right before continuing with the original dialog flow (independent of the setting of **Continuation** within the Destination section).

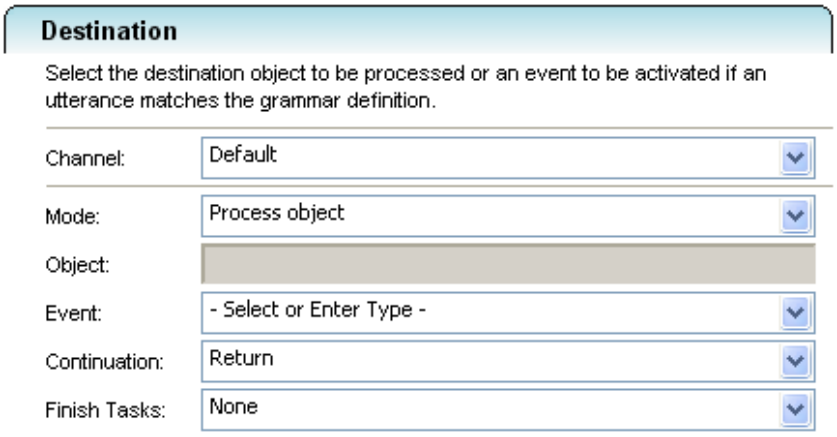
Note: Configurations made in the **Event Handling**, **Navigation** and **Tuning** sections are only interpreted if you make use of the confirmation functionality of the Hyperlink object. When simply specifying the **Destination** and **Activation Grammar**, event handling, navigation and tuning will be ignored. So, in particular, a confidence level for a specific hyperlink command cannot be defined by setting the corresponding tuning parameter in the **Tuning** section. Use the **Weight** field in the **Activation Grammar** section to increase or decrease the prominence of a specific Hyperlink object in the voice or video channel.

Copyright © 2001-2011 Voxeo Germany GmbH

Note: The embedded version of the Hyperlink object – as it occurs within the **Custom Navigation** definition of the objects Module, Input, Menu, Confirmation, List, Plug-In, and Hyperlink itself – is structurally simpler and only contains an equivalent of the **Destination** section and the **Activation Grammar** section.

Destination

The **Destination** section defines the destination of the hyperlink. There are two different modes of using a Hyperlink object. One is to process a specific object whenever the Hyperlink object is activated. The other is to activate a specific event, which can then initiate other processing.



Select *Process object* (the default) in the **Mode** field, to process a specific object when the hyperlink is activated; then specify the object to be processed by linking it in the **Object** field.

Select *Activate event* in the **Mode** field to activate an event instead; in that case, pick an event from the **Event** field or type in a custom event name. When typing in a custom event, you can make use of the "dot" notation ("myevents.event1") to work with a hierarchy of events.
When rendering the current dialog step, the server will start looking for a corresponding event handler, first in the current object, then in parent Module objects. If a handler with the same name, or, in case the event uses the "dot" notation, a more generic handler is found, it will be rendered into the VoiceXML document. If no matching handler can be found, though, a default handler gets rendered that simply logs the activation of the event to Infostore and a corresponding log message is written to the processing log files. The effect of this "silent" handler is that the caller utterance is simply ignored. If you do not want the server to render this default handler, use the `<suppressedDefaultHandlers>` element in the *MPDrivers.xml* file to list names of events that default handlers should not be rendered for.

Note: If **Mode** is set to *Process object* and no destination object is defined, or **Mode** is set to *Activate event* but no event is defined, then the hyperlink is ignored altogether and processing continues normally. A corresponding warning message in the processing log files will point to this.

The event *Caller - Help* is particularly useful in many applications, since it allows designers to activate help (as defined in event handling) using custom grammars. Note that the help event should always be activated by explicitly defining a hyperlink with a custom grammar, in order to have control over the utterances that activate the help event. You can make use of a “universal” grammar for this event, though, by

Copyright © 2001-2011 Voxeo Germany GmbH

setting the tuning parameter *Input – Universal Commands* to *help* (which is the default value for this parameter). In this case, the media platform provides a generic grammar for this command (usually containing utterances like *help* or *I need help*). For more information on setting this tuning parameter, refer to *Tuning* in this *Object Reference*.

The **Continuation** drop-down list indicates how to proceed with the dialog flow after the hyperlink has been processed. This setting only applies if the mode is set to *Process object*. If the mode is set to *Activate event*, the corresponding event handler will govern how the dialog proceeds.
The **Continuation** drop-down list has three options:

- *Return (default)*
  The original object in which the hyperlink was activated is processed again (increasing the object occurrence counter), and the prompt defined in this object is replayed. This gives the caller some orientation when coming back from the hyperlink subdialog.
  This option is commonly used when callers interrupt the dialog for additional information or instructions before proceeding.
- *Proceed*
  The object in which the hyperlink has been activated is skipped when returning from the hyperlink subdialog, and the application proceeds with the next object in the original dialog flow. If no subsequent object is defined, the call terminates.
  If you want to define a command that simply skips the current object, use an Output object as destination. You can then either define a prompt like *Okay, I'm skipping this question*, or not define any prompt in the Output object. In the latter case the dialog proceeds silently with the next object in the dialog flow.
- *Do not return*
  The application does not return to the object in which the hyperlink has been activated. This option is typically used e.g. when defining a hyperlink that jumps back to a main menu from anywhere in an application. Note that if the destination defined does not explicitly specify the dialog flow any further (e.g. when using an Output object as the destination), the dialog terminates with this option.

Note: When reprocessing the object in which the hyperlink has been activated, i.e. **Continuation** is set to *Return*, the occurrence level of this object is incremented. Be aware that this affects the selection of the prompt to play, as defined in the **Initial** section of the object where the hyperlink was activated.

The **Channel** field allows you to activate the hyperlink only for a specific channel. If you want the hyperlink to be active in all channels, leave it at *Default*.

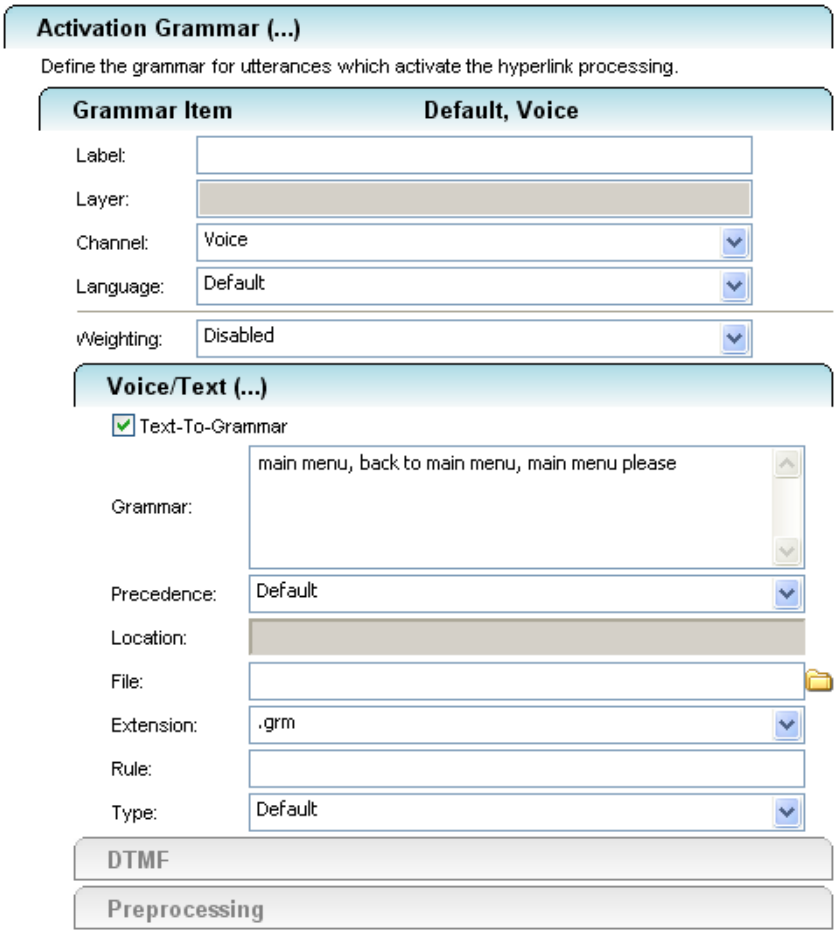
The **Finish Tasks** field defines whether currently active business tasks are to be finished or not when the hyperlink is processed. It can be set to *None* (the default), which means that no task will be finished, *Active*, which means that only the currently active task will be finished, or *All*, which means that all tasks, active or inactive, will be finished. Note that this will only happen when the hyperlink is really processed, i.e. it will particularly not happen if a confirmed hyperlink is denied by the caller. In case one or more tasks get finished, then they will automatically be marked with status *Caller Abort*, which means they will be logged as *incomplete* to Infostore.

Note: This setting is only applicable if **Mode** is set to *Process object*. If **Mode** is set to *Activate event*, **Finish Tasks** has no effect.

Copyright © 2001-2011 Voxeo Germany GmbH

Activation Grammar

The **Activation Grammar** specifies the input a caller can give to activate the Hyperlink object. This grammar does not need to fill any slots. The grammar is mandatory in all channels. If not defined, or all Grammar items are ruled out at call time due to layer conditions, the hyperlink will not be active.



In the text channel, the grammar needs to be defined using the TTG format. The activation grammar will be shown as the access key of the option the USSD browser will generate for this hyperlink. See Chapter 10 – *How to Support Multiple Phone Channels* in the *Design Guide* for examples of how the display of a text application could look like. Therefore, define some digit as the grammar for this hyperlink, e.g. *0* (zero). If you do not want to define the access key, but instead want to rely on auto-numbering on the USSD browser side, use question mark “?” as a placeholder in the grammar definition.
In the Web channel, the grammar will not be processed at all, only the presentation output will be used to display the hyperlink. Nevertheless, a grammar definition is mandatory in all channels. So either re-use the text channel grammar definition in case your application should support both channels, or use question mark “?” as a placeholder definition in case you only define a Web grammar.

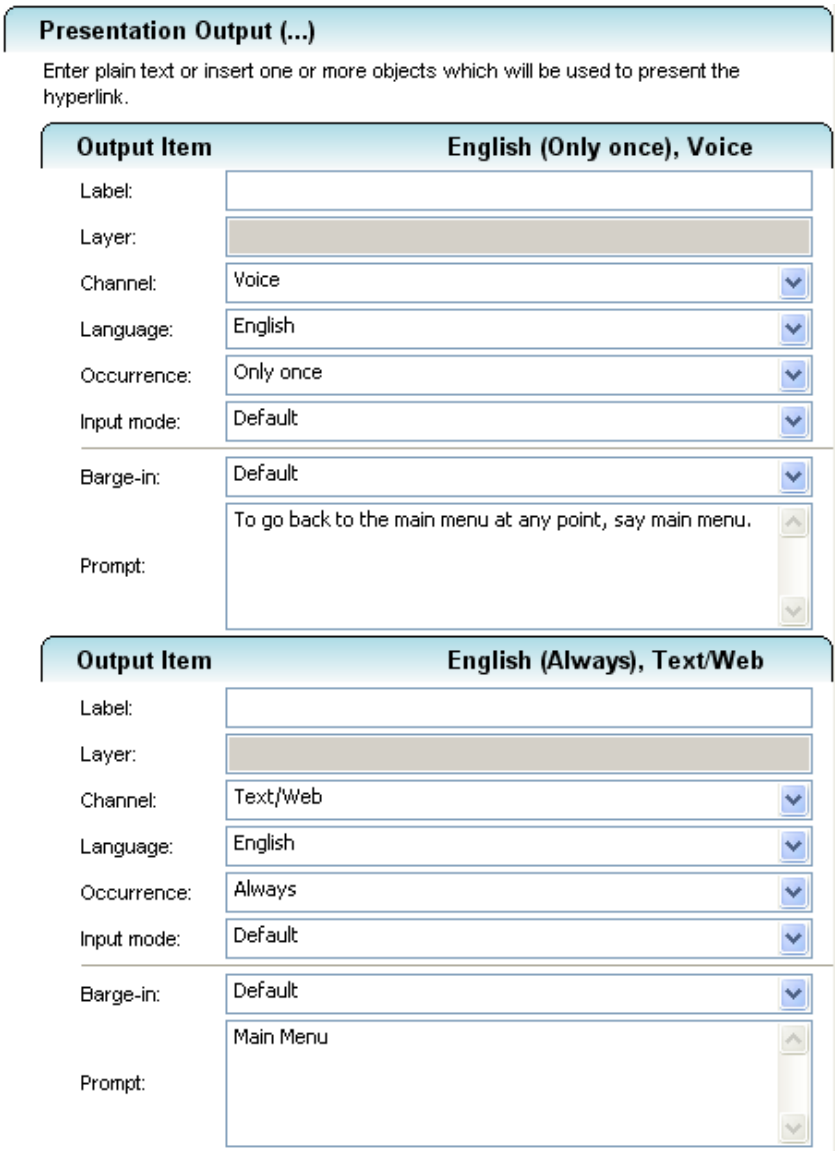
For further details on grammar definition refer to the Grammar object in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Presentation Output

The presentation output is used to present the hyperlink to the caller. In the voice and video channel, a hyperlink is fully defined with the destination and the activation grammar. Using the presentation output is optional here. You can either tell the caller about the presence of the hyperlink inside some general introduction prompt at the beginning of the dialog (e.g. in the main menu), or you use the hyperlink's presentation output to do this. A typical output could be "*To go back to the main menu at any point, say main menu*". It is recommended to set the occurrence level of this output to *Only once*, so that this prompt is only played once per call. Otherwise it will be played after the main prompt of each object in which the hyperlink is active.

In the text and Web channel, the presentation output is required, as it is used to display the hyperlink on the mobile device, so that the caller can activate it. A typical output for a main menu hyperlink could be "*Main Menu*". This would be displayed on the mobile device as an option. In the text channel, the option would be associated with the access key coming from the activation grammar. The occurrence level setting should be left at *Always*, so that a generic hyperlink is really displayed in all input states where it is active. If the presentation output is missing, the caller cannot activate the hyperlink.



Copyright © 2001-2011 Voxeo Germany GmbH

The hyperlinks are displayed at the end of each page in the text and Web channel, and played at the end of the main prompt of an object in the voice and video channel. If more than one hyperlink is active, the order in which they are presented depends on some server-internal order. If you want to define yourself in which order the hyperlinks are presented, use the **Label** field of the presentation output as a criterion for sorting. Of all hyperlinks active in an input state, the server applies alphanumerical sorting on the labels of all hyperlinks. As an example, define “01” as the presentation output label of one hyperlink, and “02” as that of another one. Finally, the server will present the “01” hyperlink first, then the “02” one.
Use this mechanism also in standard navigation commands, to define the order between custom and standard navigation.

For further details on output definitions refer to the Output object in this *Object Reference*.

Confirmation Request

The **Confirmation Request** is used in conjunction with the confirmation grammar to request an explicit confirmation from the caller before activating the Hyperlink object. This is useful for hyperlinks the activation of which would be annoying in case of misrecognition by the media platform in voice or video applications, e.g. a “goodbye” hyperlink that would end the call.

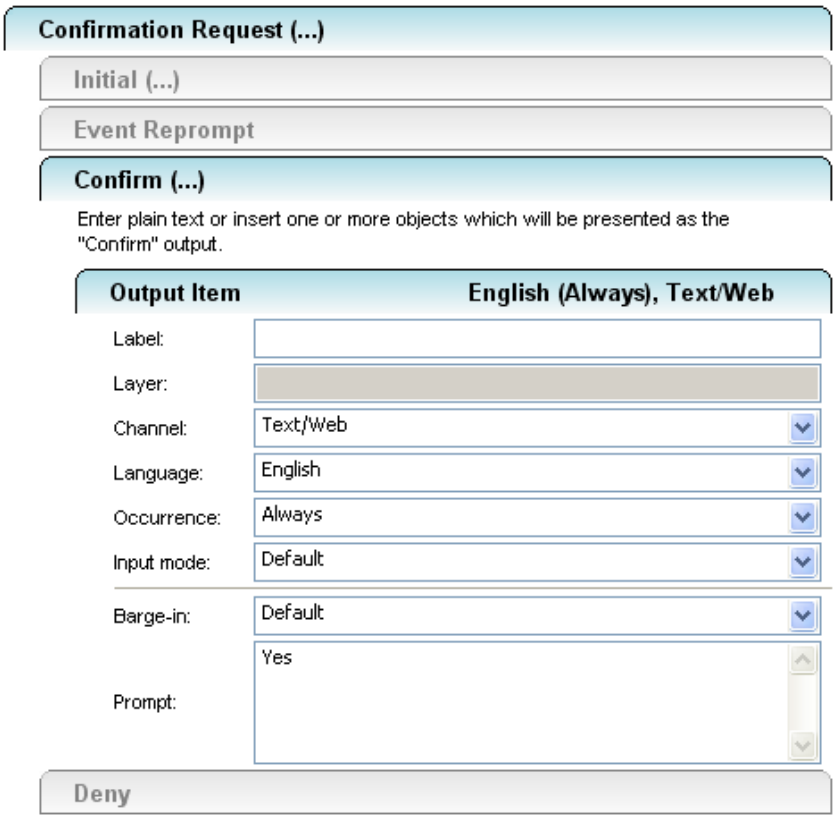
Using the confirmation functionality is optional. If not needed, leave the **Confirmation Request** and **Confirmation Grammar** empty. In the **Message** section, you can still define the *Activation* and *Return* outputs, though.



For more information on the **Initial** and **Event Reprompt** sections refer to the Input object in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

In addition to the confirmation request, a **Confirm** and **Deny** output can be defined. These are optional outputs in case of the voice and video channel; they could contain prompts that tell the caller what to say or press to confirm or deny the activation of the hyperlink, respectively. In the text and Web channel, these outputs are mandatory. Use them to define the output that should be displayed for the options to confirm and deny the hyperlink.

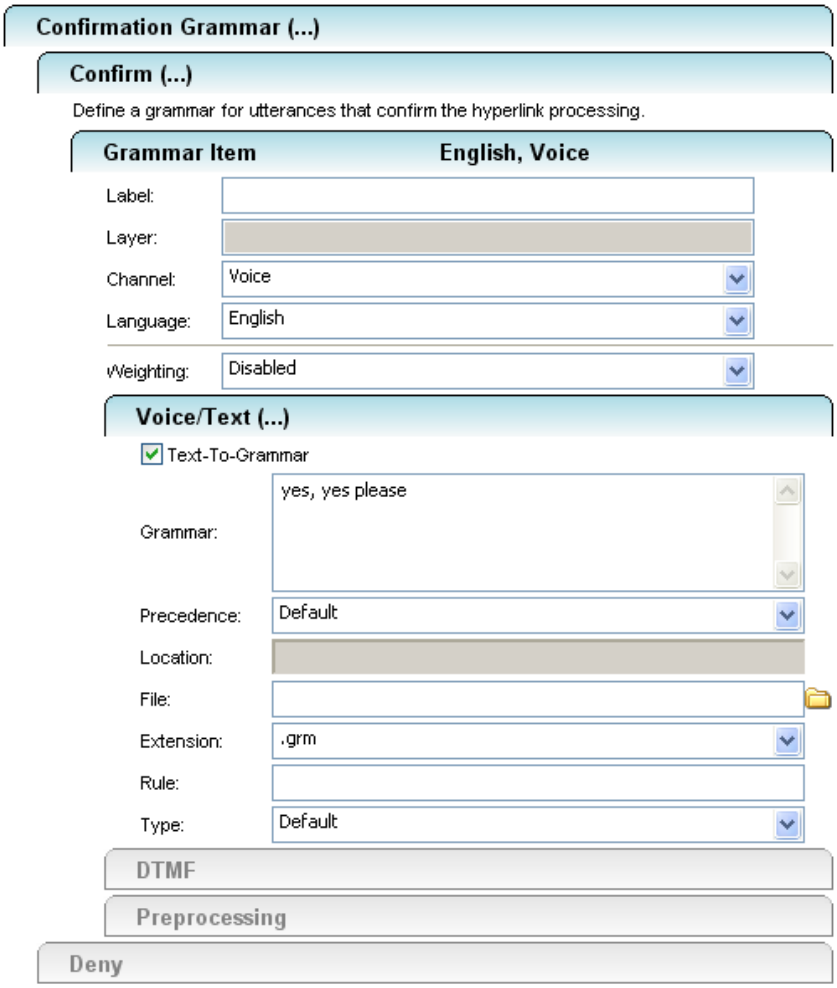


Confirmation Grammar

The **Confirmation Grammar** defines two separate grammars that indicate the input the caller can make to confirm processing of the Hyperlink object (in the **Confirm** section) or deny processing of the Hyperlink object (in the **Deny** section). Neither of these two grammars needs to fill any slots.

In the text channel, the confirmation grammar is used to define the access keys for the *confirm* and *deny* options the USSD browser will generate. In the Web channel, the grammar is conceptually not required. But since the grammar is mandatory, you can use the question mark “?” placeholder as the grammar, or use the definition of the text channel.

Copyright © 2001-2011 Voxeo Germany GmbH

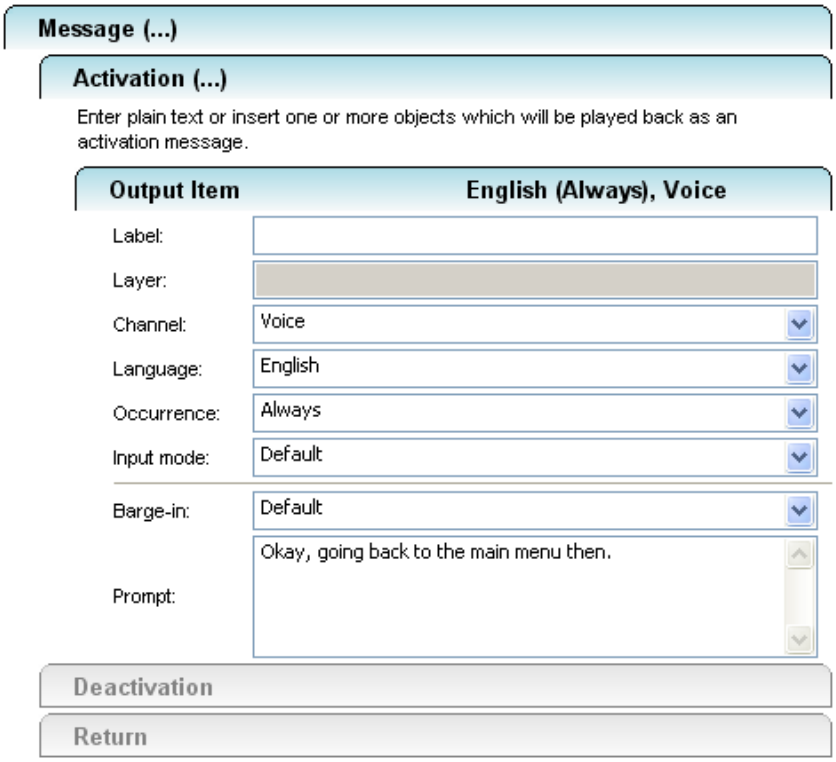


For further details on grammar definition refer to the Grammar object in this *Object Reference*.

Message

The **Message** section defines optional output for various purposes:

Copyright © 2001-2011 Voxeo Germany GmbH



Activation defines an optional output that is presented after the caller has explicitly confirmed the hyperlink, if applicable, and before the hyperlink destination object is processed.

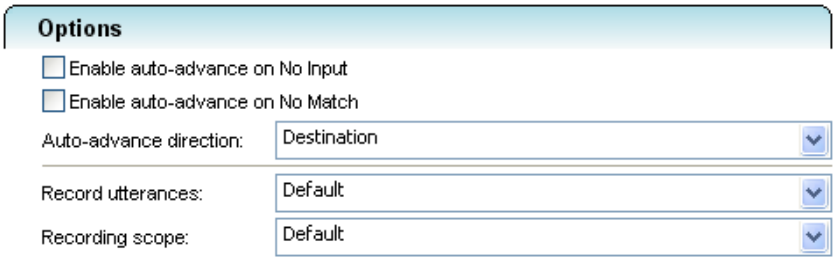
Deactivation defines an optional output that is presented if the caller explicitly denies processing of the hyperlink destination object. Note that the deactivation output can only be played if both a confirmation output and a confirmation grammar are defined.

Return defines an optional output that is presented after the hyperlink destination object has been processed and before processing returns to the normal dialog flow. Note that the return output is only played if *Return* or *Proceed* is selected in the **Continuation** field in the **Destination** section. The return output can also be used if the Hyperlink object is not a confirmed one, i.e. without having the confirmation request and grammar defined.

For further details on output definitions refer to the Output object in this *Object Reference*.

Options

The **Options** section of the Hyperlink object specifies five processing options.



Copyright © 2001-2011 Voxeo Germany GmbH

For hyperlinks that have a confirmation request defined, **Enable auto-advance on No Input** enables auto-advance in case the caller does not respond to the confirmation request. If the check box is selected, a No Input event is interpreted either as an implicit ("silent") confirmation or denial, depending on the setting of **Auto-advance direction** (see below). By default, the check box is clear. This option is only supported in the voice and video channel.

Enable auto-advance on No Match enables auto-advance in case the caller says something that does not match the confirm or deny grammars. If the respective check box is selected, a No Match event is interpreted either as an implicit confirmation or denial, depending on the setting of **Auto-advance direction** (see below). By default, the check box is clear. This option is only supported in the voice and video channel.

Auto-advance direction defines whether or not to process the hyperlink destination object if one of the auto-advance options is activated:
Destination (the default) indicates that in cases of auto-advance, the destination object is processed.
Origin indicates that in cases of auto-advance, the destination object is not processed, and the dialog returns to the dialog flow, reprocessing the current object (regardless of which option is selected in the **Continuation** field in the **Destination** section).

Record utterances specifies whether utterances made in this object are to be recorded (*True*) or not (*False*), given that utterance recording is switched on for this session. *Always* indicates that utterances in this object are always recorded when utterance recording is enabled on the service, regardless of statistical filtering. By default the value is *Default*, indicating that the current value of the corresponding dialog context setting RECORDUTTERANCES is used.

Recording scope defines the scope in which recordings will be made. If set to *Recognition*, only those utterances will be recorded that were successfully recognized (including the activation of hyperlinks). If set to *No Match*, only those utterances will be recorded that triggered a No Match event. If set to *All*, any recording will be stored. If set to *Default*, the current value of the corresponding dialog context setting RECORDINGSCOPE is used. By default the value is *Default*.

Due to the nature of recordings, the feature of recording utterances is only available in the voice and video channel. For more information on this refer to Chapter 6 – *Recording of Utterances* in the *Deployment Guide*.

VoiceObjectsXML Definition

The Hyperlink object is represented by the VoiceObjectsXML element `<hyperlink>`. It has thirteen attributes and seventeen groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

Note that the `label`, `layer` and `channel` attributes may only be used when the hyperlink is embedded in `<customNavigation>`, while the children required for confirmation as well as the attributes `recordUtterances` and `recordingScope` may only be used if the hyperlink is defined as an autonomous object.

Hyperlink

Attributes

- **`mode`**
  Defines whether to process an object or to activate an event. Legal values are *object* and *event*. If not specified, defaults to *object*.
- **`object`**
  Defines the object to be processed if the hyperlink is activated.

Copyright © 2001-2011 Voxeo Germany GmbH

- **`event`**
  Defines the event to be activated if the hyperlink is triggered. Can be any internal VoiceXML event (like *help*, *cancel*, *noMatch* etc.), or an arbitrary custom event name.
- **`continuation`**
  Defines the continuation mode. Legal values are *return*, *noReturn*, *proceed*. If not specified, defaults to *return*.
- **`label`**
  A text string providing a name for the hyperlink.
  This attribute must only be used if the hyperlink is directly embedded inside `<customNavigation>`.
- **`layer`**
  Defines the layer for the hyperlink. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form "Layer=State" or "Layer!=State" where "State" is the label of a state for the layer "Layer".
  This attribute must only be used if the hyperlink is directly embedded inside `<customNavigation>`.
- **`channel`**
  Defines the channel(s) for which this hyperlink is valid. Can be *default*, *voice*, *video*, *text*, *web*, *voiceVideo*, or *textWeb*. If not specified, defaults to *default*.
  This attribute must only be used if the hyperlink is directly embedded inside `<customNavigation>`.
- **`autoAdvanceNoInput`**
  Defines whether the dialog should automatically advance in case of a No Input event. Either *true* or *false*. If not specified, defaults to *false*.
  This attribute must only be used if the hyperlink is autonomous.
- **`autoAdvanceNoMatch`**
  Defines whether the dialog should automatically advance in case of a No Match event. Either *true* or *false*. If not specified, defaults to *false*.
  This attribute must only be used if the hyperlink is autonomous.
- **`autoAdvanceDirection`**
  Defines the direction of an auto advance. Can be either *destination* or *origin*. If not specified, defaults to *destination*.
  This attribute must only be used if the hyperlink is autonomous.
- **`recordUtterances`**
  Defines whether to use utterance recording. One of *true*, *false*, *always*, *default*. If not specified, defaults to *default*.
- **`recordingScope`**
  Defines the scope for utterance recording. One of *all*, *noMatch*, *recognition*, *default*. If not specified, defaults to *default*.
- **`finishTasks`**
  Defines whether business tasks are to be finished or not. One of *none*, *active*, *all*. If not specified, defaults to *none*.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or

Copyright © 2001-2011 Voxeo Germany GmbH

<script usage=”precondition”>
Defines the precondition for the Hyperlink object.

- **<sequence usage=”preprocessing”>**
  Defines the preprocessing sequence for the Hyperlink object.
- **<grammar usage=”activation”>**
  Defines the activation grammar for the Hyperlink object.
- **<output usage=”confirmation” type=”initial”>**
  Defines the “initial” section of the confirmation message.
- **<output usage=”confirmation” type=”reprompt”>**
  Defines the “event reprompt” section of the confirmation message.
- **<output usage=”presentation”>**
  Defines the presentation for the Hyperlink object.
- **<grammar usage=”confirm”>**
  Defines the grammar used to confirm the processing of the Hyperlink object.
- **<grammar usage=”deny”>**
  Defines the grammar used to deny the processing of the Hyperlink object.
- **<output usage=”activation”>**
  Defines the activation message.
- **<output usage=”deactivation”>**
  Defines the deactivation message.
- **<output usage=”return”>**
  Defines the return message.
- **<output usage=”confirmPresentation”>**
  Defines the confirm presentation.
- **<output usage=”denyPresentation”>**
  Defines the deny presentation.
- **<eventHandling>**
  Defines the event handling for the Hyperlink object.
- **<customNavigation>**
  Defines the custom navigation settings for the Hyperlink object.
- **<tuning>**
  Defines the tuning settings for the Hyperlink object.
- **<sequence usage=”postprocessing”>**
  Defines the postprocessing sequence for the Hyperlink object.

Example

```
<hyperlink object="#Main menu">
  <grammar>
    <grammarItem language="en-US">
      <grammarDefinition mode="voice">main menu</grammarDefinition>
    </grammarItem>
  </grammar>
</hyperlink>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
<hyperlink mode=”event” event=”help”>
  <grammar>
    <grammarItem language="en-US">
      <grammarDefinition mode="voice">help</grammarDefinition>
    </grammarItem>
  </grammar>
</hyperlink>


<hyperlink name="Goodbye" object="#Hang up" continuation="return"
autoAdvanceDirection="origin" autoAdvanceNoInput="true"
autoAdvanceNoMatch="true">
  <grammar usage="activation">
    <grammarItem>
      <grammarDefinition mode="voice">hang up, goodbye
      </grammarDefinition>
    </grammarItem>
  </grammar>
  <output usage="confirmation">
    <outputItem>
      <text>Are you sure you want to end this call?</text>
    </outputItem>
  </output>
  <grammar usage="confirm">
    <grammarItem>
      <grammarDefinition mode="voice">yes, sure, yes i am
      </grammarDefinition>
    </grammarItem>
  </grammar>
  <grammar usage="deny">
    <grammarItem>
      <grammarDefinition mode="voice">no, back, stop, no i'm not
      </grammarDefinition>
    </grammarItem>
  </grammar>
  <output usage="deactivation">
    <outputItem>
      <text>OK, we will continue.</text>
    </outputItem>
  </output>
</hyperlink>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The following table contains all object types that can reference a Hyperlink object. In general, Hyperlink objects can be used in all objects that offer navigation.

Icon	Object Name	Use Case Example
	Module	Most commonly, a Hyperlink object is referenced in the **Navigation** definition of a Module object.
	Input	A Hyperlink object can be referenced in the **Navigation** definition of an Input object.
	Menu	A Hyperlink object can be referenced in the **Navigation** definition of a Menu object.
	Confirmation	A Hyperlink object can be referenced in the **Navigation** definition of a Confirmation object.
	List	A Hyperlink object can be referenced in the **Navigation** definition of a List object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Hyperlink object should indicate the target of the hyperlink. The short description should also provide an indication which utterances activate the hyperlink, and whether the hyperlink will return to the current position in the dialog or not. The table below lists three examples:

Name	Description
 Main Menu	Jumps back to the main menu. Activated only by the command *main menu*. Does not return.
 Switch to female Persona	Switches the active Persona to the female one. Activated by a variety of commands such as *female Persona*, *switch to female speaker*, etc. Returns to current position after switching and replays prompt.
 Pause application	Enters the Pause object. Activated by utterances such as *short break* or *pause application*. Returns to current position when caller continues and replays prompt.

Copyright © 2001-2011 Voxeo Germany GmbH

If

Overview

The If object represents a standard IF-THEN-ELSE construct as in any standard programming or scripting language. It is typically used within the dialog flow to fork it into two separate branches depending on certain business rules.

Usage examples for the If object are:

- Processing an optional introductory tour if the caller chooses to hear it.
- Switching between two different subdialogs depending on which of two options the caller has selected.
- Switching between a regular menu used on weekdays and a recorded announcement used on weekends.

The Object Definition below covers the configuration of the If object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML, refer to the VoiceObjectsXML Definition paragraph.

The If object belongs to the object category Logic.

Dialog Flow Scenario

The dialog flow example below presents the use of the If object within a voice shopping portal. It branches the dialog flow into two different welcome Sequence objects depending on the application visit status of the particular caller. The following dialog flow demonstrates the THEN branch where the visit counter is evaluated as greater than zero.

Object	Dialog Flow
	< Connect to the back-end system to load the visit counter of the caller based on the supplied ANI (Automated Number Identification) >
	[ Music jingle] *Hello and welcome to the voice shopping portal.*
	…< Visit counter greater than 0 > [true]
	< Frequent Visitor >
	*I'm pleased to assist you again. …*
	*What can I do for you today? …*

The next dialog flow example demonstrates the ELSE branch of the same If object. Here, the visit counter for the particular caller is equal to zero and the If condition evaluates to false. The introduction offers a quick tour through the voice shopping portal to provide the caller with an overview of the service for better orientation. After the orientation tour the dialog flow proceeds with the main menu.

Copyright © 2001-2011 Voxeo Germany GmbH

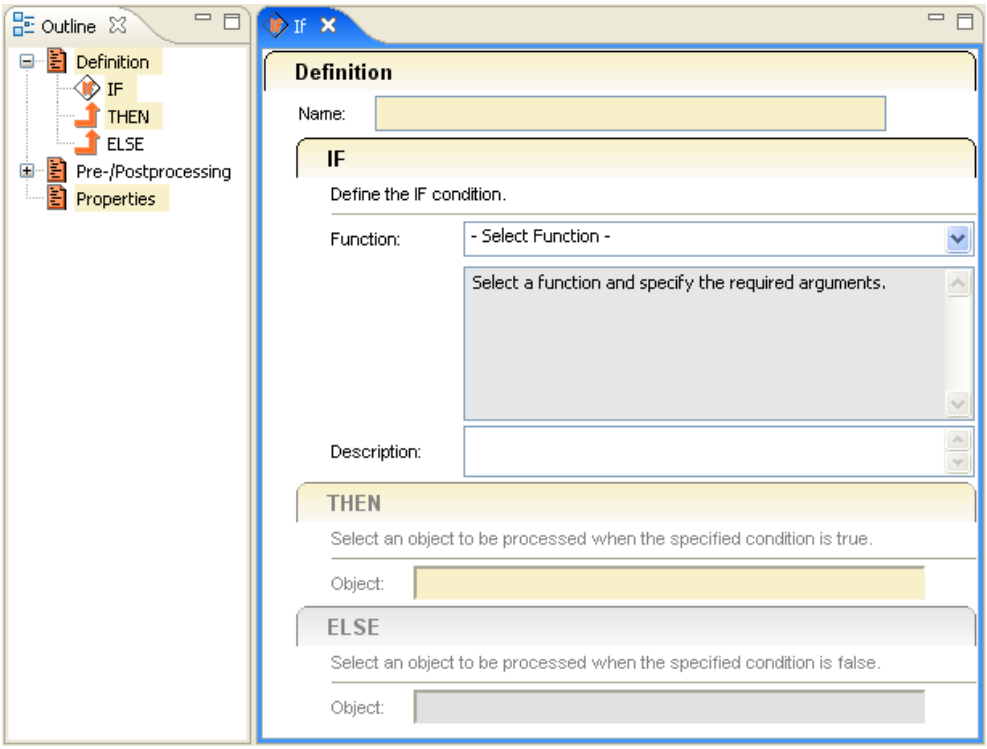
Object	Dialog Flow
	< Connect to the back-end system to load the visit counter of the caller based on the supplied ANI (Automated Number Identification) >
	[ Music jingle] *Hello and welcome to the voice shopping portal.*
	…< Visit counter greater than 0 > [false]
	< First Time Visitor >
	*Since this is your first visit to the voice shopping portal, I'd like to give you a quick tour of how to use the system.* *If you want to skip the tour, say "main menu" now.* < Auto-advance with dialog flow after 2 seconds >
	*OK, let's get started with the orientation tour. In the music shop you will find …*
	*Main menu! Which shop would you like to visit? …*

Object Definition

The **Definition** of the If object provides the following sections:

- IF
  To define the branching condition.
- THEN
  To define the object that is processed if the branching condition is true.
- ELSE
  To define the object that is processed if the branching condition is false.

Copyright © 2001-2011 Voxeo Germany GmbH



For further details regarding additional object configuration refer to *Precondition* and *Properties* in this *Object Reference*.

IF

The **IF** section specifies the Boolean condition that is evaluated to determine whether the THEN or the ELSE portion of the object is processed. Either define an embedded condition by selecting an appropriate operator from the **Function** drop-down list and specifying the required arguments, or link a Variable, Expression, Collection, Script, or Layer object that returns a Boolean value. For more information on the use of functions, see the Expression object in this *Object Reference*.

THEN

The **THEN** section defines the object that is processed if the IF condition evaluates to *true*. The THEN object is optional if an ELSE object is defined.

ELSE

The **ELSE** section defines the object that is processed if the IF condition evaluates to *false*. The ELSE object is optional if a THEN object is defined.

VoiceObjectsXML Definition

The If object is represented by the VoiceObjectsXML element `<if>`. It has five groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<if>` element uses the embedded **`<ifItem>`** and **`<else>`** elements.

Copyright © 2001-2011 Voxeo Germany GmbH

If

Children

- **<expression usage="precondition">** or
  <variable usage="precondition"> or
  <collection usage="precondition"> or
  <script usage="precondition">
  Defines the precondition for the If object.
- **<sequence usage="preprocessing">**
  Defines the preprocessing sequence for the If object.
- **<ifItem>**
  Defines the condition and the object to be processed for the "then" case.
- **<else>**
  Defines the object to be processed for the "else" case.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the If object.

Example

```
<if>
  <ifItem>
    <expression usage="condition" function="equal">
    <!-- Checks whether the number is even -->
      <argument>
        <expression function="mod">
          <argument object="#Number"/>
          <argument><value> 2 </value></argument>
        </expression>
      </argument>
      <argument><value> 0 </value></argument>
    </expression>
    <output>
      <outputItem><text>The number is even.</text></outputItem>
    </output>
  </ifItem>
  <else>
    <output usage="else">
      <outputItem><text>The number is odd.</text></outputItem>
    </output>
  </else>
</if>
```

Copyright © 2001-2011 Voxeo Germany GmbH

IfItem

Children

- **<expression usage="condition">** or
  <variable usage="condition"> or
  <collection usage="condition"> or
  <script usage="condition">
  Defines an embedded object as condition.
- One of the following:
  <module>, <input>, <output>, <sequence>, <menu>, <confirmation>, <list>, <osdm>, <plugin>, <connector>, <script>, <log>, <if>, <case>, <goto>, <loop>, <expression>, <recording>, <transfer>, <pause>, <exit>
  Defines the object to be processed if the condition evaluates to true.

Example

```
<ifItem>
  <expression usage="condition" link="#isLeapYear"/>
  <output>
    <outputItem><text>The year is a leap year.</text></outputItem>
  </output>
</ifItem>
```

Else

Children

- One of the following:
  <module>, <input>, <output>, <sequence>, <menu>, <confirmation>, <list>, <osdm>, <plugin>, <connector>, <script>, <log>, <if>, <case>, <goto>, <loop>, <expression>, <recording>, <transfer>, <pause>, <exit>
  Defines the object to be processed in the "else" case (i.e. when the condition evaluates to false).

Example

```
<else>
  <output>
    <outputItem><text>The year is not a leap year.</text></outputItem>
  </output>
</else>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The following table contains all the object types that can refer to an If object:

Icon	Object Name	Use Case Example
	Module	An If object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, the If object is used within a Sequence object in a dialog flow.
	Menu	An If object can be used as the destination object within a Menu item inside a Menu object.
	Confirmation	An If object can be used as the destination object within a Correction item inside a Confirmation object.
	If	An If object can be used within either the THEN or the ELSE item of another If object.
	Case	An If object can be used in any WHEN item of a Case object.
	Loop	An If object can be used within the embedded sequence of a Loop object.
	Goto	An If object can be referenced via a Goto object.
	Hyperlink	An If object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of an If object should indicate the condition that is evaluated, preferably in the form of a Yes/No question. The short description should contain a brief explanation of the different cases that may occur. The table below lists three examples:

Name	Description
 Caller wants tour	Processes the service tour if desired by the caller (answered *yes*).
 Call during weekdays	Switches between the normal menu used on weekdays and a recorded announcement used for weekends.
 Account balance above $10,000	Switches between service level for normal customers (account balance below $10,000) and premium customers (account balance above $10,000).

Copyright © 2001-2011 Voxeo Germany GmbH

Input

Overview

The Input object is a configurable dialog component that plays an Output to request information from the caller and assigns the caller's input to one or more Variable or Layer objects. The input can be either DTMF or recognized voice input in the voice and video channel, or text in the text and Web channel.

The Object Definition below covers the configuration of the Input object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML, refer to the VoiceObjectsXML Definition paragraph.

The Input object belongs to the object category Components.

Dialog Flow Scenario

The first dialog flow example describes a simple input dialog requesting a phone number.

Object – Caller	Dialog Flow
	*Please say your 10-digit phone number, or enter it using your telephone keypad.*
 Caller	*703 432 7564*

The second dialog flow example is taken from a flight booking application.

Object – Caller	Dialog Flow
	*Would you like arrival or departure information?*
 Caller	*Departure*

The third dialog flow example is taken from the same flight booking application, and shows a single input for multiple information items (origin and destination of a flight).

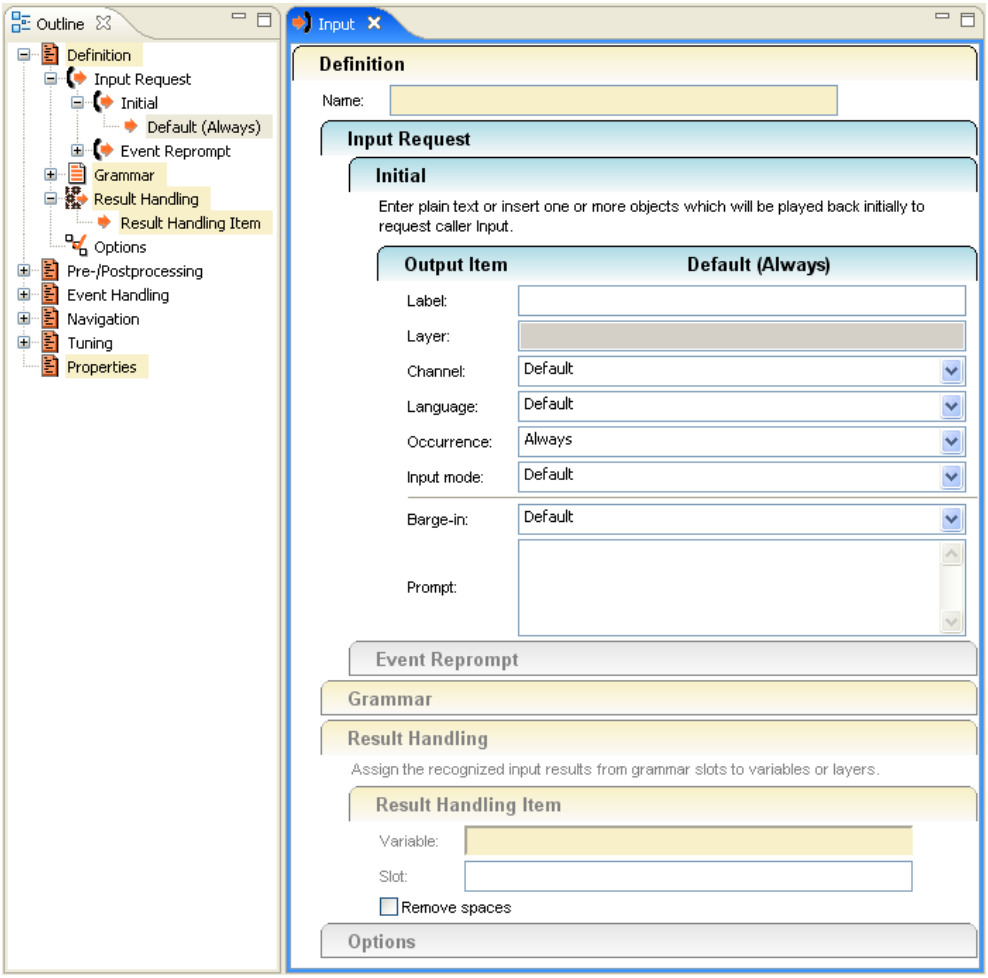
Object – Caller	Dialog Flow
	*From where to where would you like to travel?*
 Caller	*I want to fly from San Francisco to New York City.*

Copyright © 2001-2011 Voxeo Germany GmbH

Object Definition

The **Definition** of the Input object provides the following sections:

- Input Request
  To specify the outputs that ask the caller for input.
- Grammar
  To define the inputs to be accepted.
- Result Handling
  To specify where to store the caller input.
- Options
  To configure processing options.

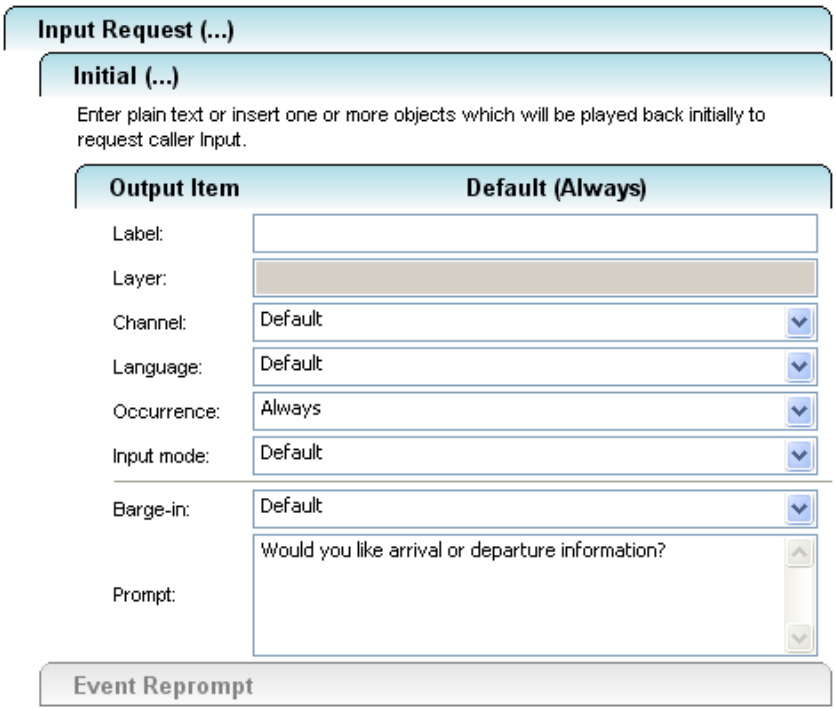


For further details regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Navigation*, *Tuning*, and *Properties* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Input Request

The **Input Request** holds an embedded Output object and is used to ask the caller for the required input specified in the **Initial** section, as well as to define modified outputs for event cases in the **Event Reprompt** section (which is only processed in the voice and video channel). The output can be defined by entering plain text or by linking any combination of text, Output, Variable, Collection, Expression, Layer, Script, Audio, Video, and Silence objects. In the voice and video channel, plain text will by synthesized by the TTS engine of the media platform (if available). In the text and Web channel, the text will be displayed on the screen of the mobile device.



For further details refer to the Output object in this *Object Reference*.

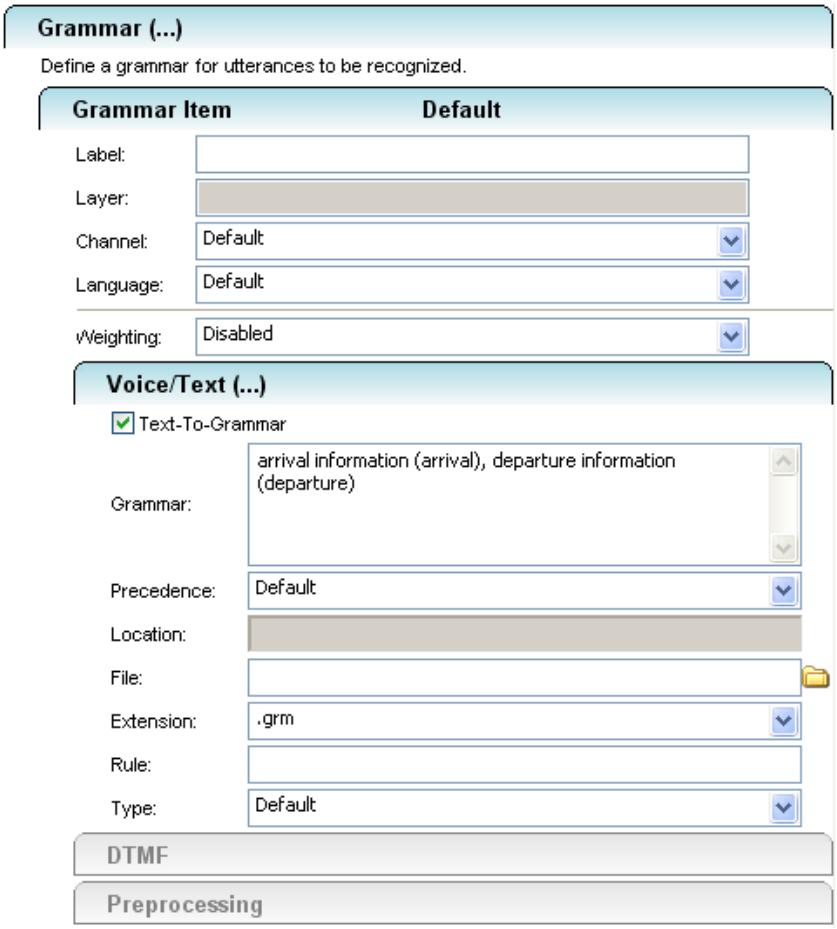
For a detailed description of how to use the reprompting mechanism, refer to Chapter 8 – *Advanced Event Handling* in the *Design Guide*.

Grammar

The **Grammar** section defines the grammar specifying the utterances that can be recognized as input. In the voice and video channel, the grammar can define voice and/or DTMF utterances and is interpreted by the speech and DTMF recognition engines of the media platform. In the text and Web channel, the grammar can be used to reduce the possible text input to a short list of alternatives.

Tip: To develop dynamic speech or DTMF grammars effectively you can use a grammar tool like *NuGram IDE*, which provides a complete set of Eclipse tools to author, test, and debug speech or DTMF recognition grammars. For further details and a free download refer to the Nu Echo Website (www.grammarserver.com) or the VoiceObjects Developer Portal (http://developers.voiceobjects.com/partners/).

Copyright © 2001-2011 Voxeo Germany GmbH



Caution: At least one Grammar item must be defined in this section to get a valid Input object definition. If no grammar is defined at all, or all Grammar items are disabled at call time due to layer evaluation, the Input object will not work.

In the example above, possible inputs are *arrival information* and *departure information*.

In the voice and video channel, the grammar can be implemented through an embedded definition or an external file, or be dynamically generated at call time. The grammar format that needs to be used depends on the requirements of the media platform. The example here uses the VoiceObjects text-to-grammar (TTG) format, which is translated into a real grammar by the server at call time.

Caution: Some media platforms do not support slot assignments in DTMF grammars. These platforms always return the DTMF keys pressed, but no semantic interpretation. If you use *TTG* for DTMF grammars, this will be handled by the server, i.e. in this case slot assignments will work as usual. If you use external, or inline but non-TTG grammars, you need to be aware of this restriction. See Appendix A – *Media Platform Drivers* in the *Deployment Guide* to find out how your media platform handles this.

In the text and Web channel, the grammar *must* be defined as a TTG grammar in the **Voice/Text** section of the Grammar object. A TTG grammar can also be generated dynamically at call time, though. If more than one Grammar item is active at call time, only the first one will be interpreted.
Grammars are typically used in these channels to reduce the possible text input to a short list of alternatives. If the number of utterances defined in the TTG grammar is

Copyright © 2001-2011 Voxeo Germany GmbH

small enough, it is best practice to display the grammar as a set of alternatives, so that the caller can simply choose a predefined entry instead of typing it in. If the TTG grammar defines utterances and explicit slot values, the utterances are being displayed. The server applies a default selection of presentation style if no value for the tuning parameter *Presentation – Input* (see below) is defined. By default, if the number of utterances is >= 1 and <=5, the server shows the grammar as a *menu* of labeled items in the text channel, or a list of hyperlinks in the Web channel for single-slot Input objects. In case of a multi-slot Input object in the Web channel, the grammar is shown as radio buttons by default. If it is >= 6 and <= 20, the grammar is also shown as a menu of labeled items in the text channel but as a drop-down list in the Web channel. If it is above 20, the grammar is not displayed; instead the callers must type their answer manually. In this case, the server maps the caller's response to the slot values defined in the TTG grammar (if any) if the response matches one of the utterances of the grammar. If not, the response itself is used as the slot value. (Note that the mapping of utterances to slot values is case-sensitive). If you do not want to rely on this automatic decision of how to show the grammar, use the tuning parameter *Presentation – Input* to customize the behavior. This parameter can be set to one of *text* (for text fields), *password* (for text fields in password mode), *menu* (for a list of hyperlinks), *list* (for drop-down lists), or *radio* (for radio buttons) in the Web channel, and to *text* (for free text input) or *menu* (for a list of options) in the text channel. In case of multi-slot Input objects in the Web channel, a special notation allows the definition of an input presentation style per slot. This is described in the section on Result Handling.

In the Web channel, the number of characters to be entered in a *text* or *password* field can be limited. Use the tuning parameter *Presentation – Input Maximum Length* to define this limit, either for all slots, or on a per-slot basis. To define the length per slot, the value of the parameter must be a comma-separated list of numbers with slot names in parentheses. Example: If you want to limit the length for the slot named *Age* to 3 and that of a slot *PIN* to 5, define this: 3 *(Age), 5 (PIN)*. This can be defined in a Module object and inherited by Input objects, or defined locally. Note that a new definition of this parameter always overwrites *all* of an inherited value.



Note: The text and Web channels, as opposed to voice and video, allow free input of text, since no recognition needs to take place (which only works on predefined grammars). This means that the Input object conceptually does not require a grammar to be defined. But since a grammar must be defined to create a valid Input object, use the question mark symbol "?" as the TTG definition to tell the server that free input is allowed. In this case, any response coming back from the caller will be used as the slot value.

For further details on the Grammar object refer to the section on Grammar in this *Object Reference*.

For more information on the distinction between voice and DTMF grammars and on the system layer *input mode*, which can influence both prompt and grammar selection, refer to Chapter 9 – *How to Use Different Input Modes* in the *Design Guide*.

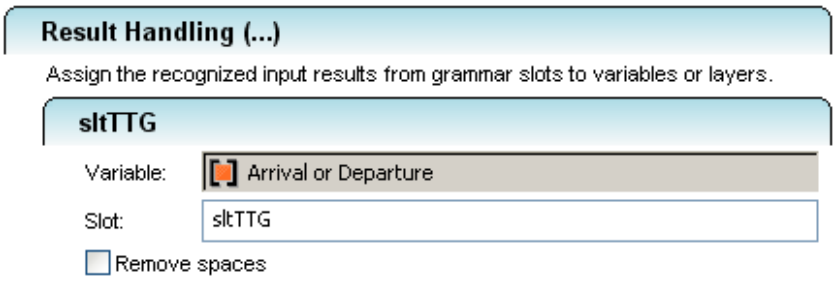


Tip: For the ability to validate free-form utterances in the text and Web channel, refer to the section *Utterance Validation* in the description of the Grammar object.

Copyright © 2001-2011 Voxeo Germany GmbH

Result Handling

In the **Result Handling** section, **Result Handling Items** are defined that assign grammar slot values to Variable or Layer objects.



Caution: At least one **Result Handling Item** must be defined. An empty result handling definition results in an invalid Input object that cannot be processed correctly by the server.

In voice and video applications, slots are the standard mechanism used in speech recognition grammars to return what is often called the semantic interpretation or representation of the caller input.

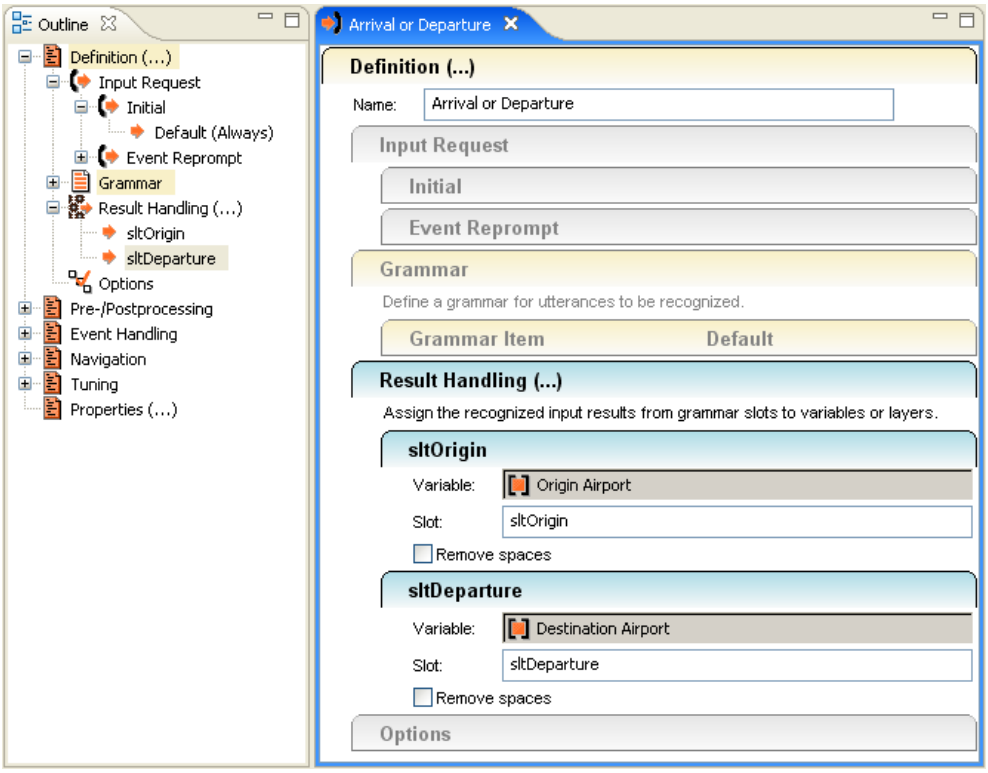
In many cases, callers have a variety of options to say essentially the same thing. In response to a question "*Is this correct?*" the caller might say "*Yes, that is correct*", "*Sure*", "*Yeah, that's right*", etc. All of these mean the same thing, namely *yes*. Thus it makes sense to map all of them to a single representation that can then be interpreted by the rest of the dialog flow. Slots provide exactly these representations. Different grammar formats define slots in different ways. For further details refer to the Grammar object in this *Object Reference*.

In addition to the slot value the grammar can define a *pronunciation value*. This value is meant to instruct the server to use either the same intonation or rhythm that the caller used or the same synonyms, in case the application reads back the input to the caller. This only applies to Variable objects, so pronunciation values cannot be set for Layer objects. The pronunciation pattern can be attached to the slot value by separating it from the actual value using the double-pipe symbol "||". If no such pattern is defined, the variable's pronunciation value will be empty. In addition, the pronunciation value is set to the empty string whenever the Variable object is part of an operation that changes its actual value, e.g. through the use of Expression objects. The pronunciation value will automatically be used as the value to format in case a Format object is applied that uses one of the predefined TTA types. In case of a custom formatting algorithm, see Appendix B – *How to Use the Formatting Bus* in the *Administration Guide*.
As an example, consider an Input object with a slot named *sltPhone* that asks for the caller's phone number. If the caller speaks the number in groups of 2 digits, the grammar can assign this grouping to the pronunciation value of *sltPhone*. If the caller for instance says "*twenty-two, sixty-three, fourty-four*", the value of the *sltPhone* slot could be *226344||22 63 44*. The part before the double-pipe symbol will become the actual variable value, whereas the part thereafter will become the pronunciation value. When reading back this number in a later output and applying the formatting type *TTA – Words/Numbers* to the Variable object, the system would read back the number in the same way, achieving an increased intelligibility towards the caller.
For more information on the pronunciation value of a Variable object, refer to Variable and Format in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

The Input object fully supports both single-slot and multi-slot inputs. In single-slot inputs, callers are asked for exactly one information item as in "*From which airport do you want to travel?*" In multi-slot inputs, callers can provide multiple information items in a single utterance, e.g. the origin and destination airport in "*I want to fly from San Francisco to Boston*". The corresponding slot values (e.g. SFO for San Francisco and BOS for Boston) are then filled into the variables specified in the **Result Handling** of the Input object.

Note: Spaces in slot values are trimmed, and space clusters (more than one space character) are reduced to one space character.



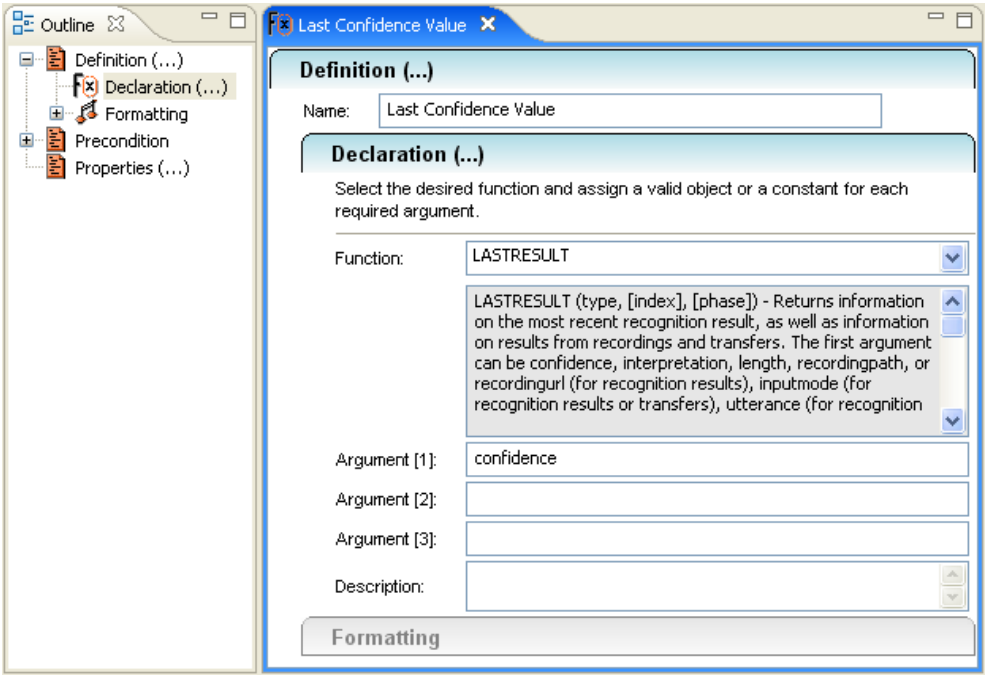
If a certain slot is not filled in the grammar by the caller's utterance (e.g. the caller said "*I want to fly to Boston*" only), the variables corresponding to the undefined slots are filled with an empty string. During the dialog flow you can check for this e.g. by using the ISNULL() function provided within the Expression object (or alternatively by just using the variable itself as a precondition; see the Variable object for a description of how variable values are interpreted as Boolean conditions). This enables you to easily build dialogs that allow callers to provide as much information at a time as they want, and that dynamically and elegantly ask for any missing information.

Note: Multi-slot inputs are not supported in the text channel. Any additional slot definition but the first one will be ignored in this channel. In the Web channel, however, multiple slots are displayed as multiple input fields. The style chosen for each slot (*text*, *menu*, or *radio*) depends on the number of utterances defined in the grammar for that slot, which is described in the section on Grammar. Using the tuning parameter *Presentation – Input*, an explicit presentation style can be defined per slot, overriding the automatic style selection applied by the server. To define a style per slot, the value of the parameter must be a comma-separated list of styles with slot names in parentheses. Example: If you want to use a text field for the slot named *Age* and a radio-button group for *Gender*, define this: *text (Age), radio (Gender)*. This can

Copyright © 2001-2011 Voxeo Germany GmbH

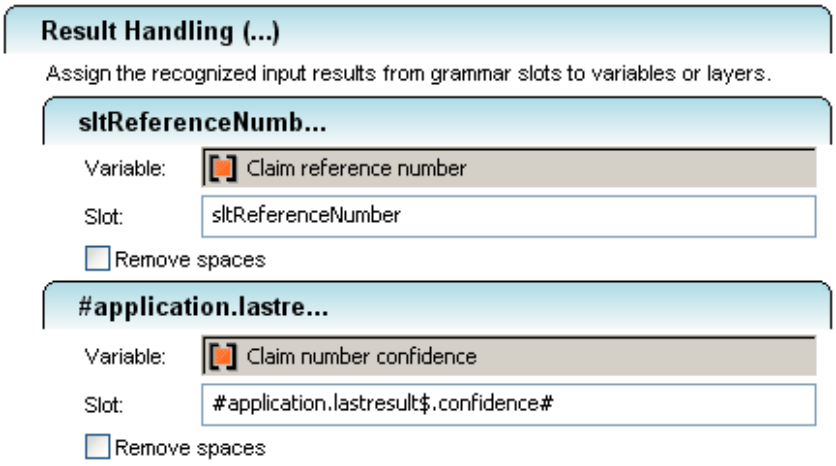
be defined in a Module object and inherited by Input objects, or defined locally. Note that a new definition of this parameter always overwrites *all* of an inherited value. By default, there is no label displayed in front of the input fields/radio button groups/drop-down lists. To be able to specify such a label, apply the tuning parameter *Presentation – Slot Label*. The value of the parameter must again be a comma-separated list of labels with slot names in parentheses. Example: If you want to show *Gender:* with a line break in front of your radio button-style slot named *Gender*, define this: *Gender:|(Gender)*. The pipe symbol "|" works the same way as in Output objects; it gets replaced with the <br/> element. Alternatively, you can also use <br/> or any other markup directly in a label.

In voice and video applications, media platforms typically return additional information regarding the most recent recognition result, such as recognition confidence or the actual caller utterance. There are two different ways to access this kind of information. The first one is to use the function LASTRESULT() within an Expression object, which returns information on last confidence, utterance, input mode, and semantic interpretation and can easily be assigned e.g. to a variable for further processing:



If, on the other hand, you are only interested in the values of a specific Input object, you can use result handling:

Copyright © 2001-2011 Voxeo Germany GmbH



The example shown here retrieves both the slot value *sltReferenceNumber* containing the recognized number as well as the recognition confidence value assigned by the media platform (a value between 0 and 1, indicating how sure the media platform is that the recognition result is correct). This confidence value may then be used e.g. to initiate a confirmation of the input if it falls below a certain threshold.

In general, media platform variables can be accessed by referencing them between hash signs (#). Typical values of interest are:

- **#application.lastresult$.confidence#**
  A number between 0 and 1, with 0 indicating lowest confidence and 1 indicating highest confidence.
- **#application.lastresult$.utterance#**
  Provides the actual string of words recognized by the media platform, as opposed to the derived slot value. For example, the slot value *123* may be returned for both the utterances *one two three* and *one hundred twenty three*.
- **#application.lastresult$.inputmode#**
  Provides the mode used by the caller to make this input; can be either *dtmf* or *voice*.

Note: In order to function correctly, an Input object needs to have at least one "normal" slot in addition to media platform value references using the "#" notation. Note also that media platform variables are not supported in the text and Web channel. The LASTRESULT function will still return values for the utterance and confidence (always 1), though.

In voice, video and Web applications, the **Slot name** must correspond to the names used in the grammar. In text applications, the slot name can be left empty.

N-Best Handling

Speech recognizers typically do not return a single hypothesis about what the caller said, but a whole array of hypotheses, all usually with different confidence values.

To access the different results, use the system function LASTRESULT(), as described within the Expression object in this *Object Reference*. The size of the array (the number of hypotheses returned) varies depending on the grammar and the setting of *Max N-Best*, a tuning parameter that can be set in the **Tuning** definition of various objects. The default value of this parameter is 1; so in order to be able to work with N-best results, set *Max N-Best* to a value greater than 1. The number of results returned

Copyright © 2001-2011 Voxeo Germany GmbH

is always between 1 and *Max N-Best*. For further details, refer to *Tuning* in this *Object Reference*.

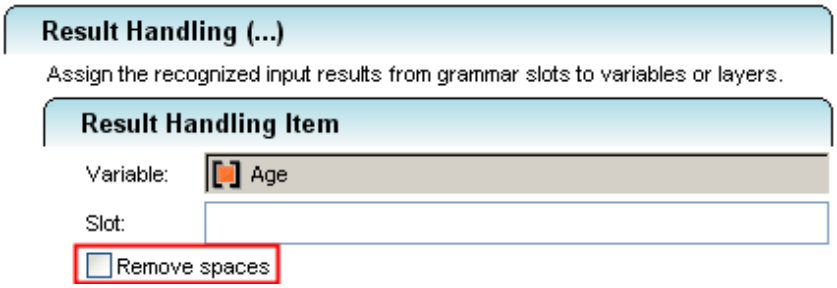
N-Best handling is not supported in the text or Web channel.

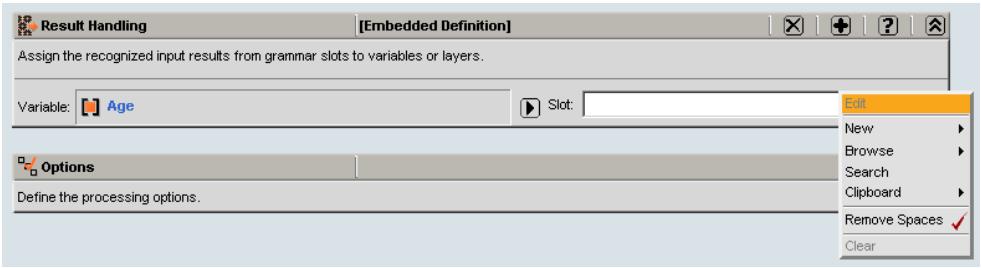
Grammars without slot assignments
Some grammars do not return named slots, e.g. the built-in grammars provided by many media platforms. To make use of such grammars, it is possible to have a single entry in the **Result Handling** without a slot name. The value returned by the grammar is then assigned to the variable specified in the **Variable** field. This single nameless slot may be combined with any number of entries using the "#" notation described above.

When using text-to-grammar style grammars (see the Grammar object), the default slot is *sltTTG* for all media platforms certified with the VoiceObjects platform (note, though, that it is possible to define a different slot name through a media platform driver if required). For reasons of convenience, you can also leave this specific slot out when defining the result handling. This is also the case in the text and Web channel, where TTG is the only format supported.

Note: The format in which utterances are returned when no named slot is present differs between the media platforms. Some platforms insert spaces between items in the utterance, while others do not.

Remove spaces from slots
In case the slot return value contains spaces which need to be removed, typically between digits in a digit sequence, select the checkbox **Remove Spaces** in the respective **Result Handling Item** section (in Desktop for Web use the context menu on the corresponding **Slot** field instead):





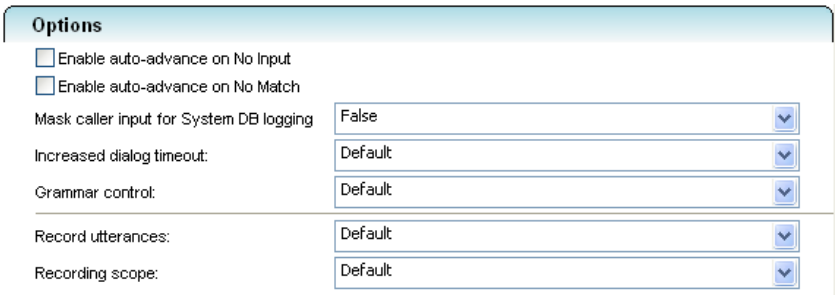
Before assigning the slot values to the Variable or Layer object of the respective result handling item, the server will remove all space characters.

Note that activating this feature in the context menu is only possible after defining a slot name or linking a Variable or Layer object.

Copyright © 2001-2011 Voxeo Germany GmbH

Options

The **Options** section of the Input editor specifies the following processing options:



Enable auto-advance on No Input enables auto-advance in case no response from the caller is detected (only supported in the voice and video channel). If the check box is selected, a No Input event finishes the processing of the Input object and proceeds to the next object in the dialog flow instead of processing the event handling specifications. By default, the check box is clear.

Enable auto-advance on No Match enables auto-advance in case the caller says something that cannot be matched with the grammar (only supported in the voice and video channel). If the check box is selected, a No Match event finishes the processing of the Input object and proceeds to the next object in the dialog flow instead of processing the event handling specifications. By default, the check box is clear.

Note: These options are for your convenience only. The same effect can be achieved by specifying a No Input or No Match event handling for level 1 and **Continuation** set to *Proceed*. If you want to proceed on the first No Input or No Match event, you may use these options. If you want to proceed at a later event occurrence level, specify appropriate event handling. Using a dedicated event handler also allows you to process any object before proceeding, e.g. play a prompt through an Output object.

Mask caller input for System DB logging encrypts the caller input before writing it to the system database via Infostore, the logging component of VoiceObjects Server. This is useful for sensitive data, like PINs, which mustn't be stored as plain text for security reasons.
The default setting *False* indicates that the data is not encrypted. *True* indicates that data is encrypted in such a way that while the original value cannot be recovered, different input values still produce different encrypted values so that subsequent analyses are possible. *Complete* indicates that the values are completely masked by a single static string. (By default the string "***MASKED***" is used; it can be configured in *components_VOServer.xml* using the `<maskUserInputComplete>` element.)

Increased Dialog Timeout specifies a timeout value that is applied for this object in case it is higher than the current dialog timeout in a session. If it is lower, the setting is ignored. It is mainly applicable in the text and Web channel, where the dialog timeout of a service should be set to a relatively low value, e.g. 40 seconds, to keep the number of concurrent sessions low, as they usually run into the server timeout due to technical reasons. Use this setting to increase the timeout temporarily for objects that are likely to require more time to respond to, e.g. when the caller has to type in an address.

Grammar Control specifies whether this Input object should support grammar-driven application control. By default the value is *Default*, indicating that the current value of the corresponding dialog context setting GRAMMARCONTROL is used.
Grammar control allows the caller both to branch out to a different part of the application bypassing the current dialog step, and to correct misrecognition from a

Copyright © 2001-2011 Voxeo Germany GmbH

previous Input object in case it was implicitly confirmed by the system in a following input state. Technically, this feature allows setting Variable/Layer objects and/or making the server process a different object. To make use of it, a grammar must be defined in addition to the currently active grammar for this Input object. The new grammar must fill a slot called *vogrammarcontrol* with instructions for the server on objects to jump to and variables/layers to fill. The instructions are defined as the value of this slot. Assignments and jump commands are separated using the semi-colon.

To assign a value to a Variable object, use the syntax *varXXX=YYY*, where *XXX* is the reference ID of an existing Variable object, and *YYY* the value. To assign the empty string to a variable, use the notation varXXX=undefined.
To set a Layer object to a different state, use the syntax *layXXX=YYY*, where *XXX* is the reference ID of an existing Layer object, and *YYY* the desired state ID.

To define jump instructions, one *gosub=XXX* and one *goto=YYY* command can be defined, none of which are mandatory. *XXX* can be one or more (separated by commas) reference IDs of objects to be processed as sub-dialogs; after processing them the server returns to the current object or processes the *goto* command, if given. YYY can be one reference ID of the object to be processed, after which processing should stop. If no *goto* command is used in the *vogrammarcontrol* slot, a *continuation* can be defined, the value of which can be *return* (the default) and *proceed*. *Proceed* makes the server go on with the next object in the flow, after having processed any variable/layer assignments and/or *gosub* instruction. *Return*, the default, instructs the server to re-process the current object. This is required to implement implicit correction.

As an example, consider the following use case:

System: *Welcome to Prime Insurance. Do you want information on car insurance, life insurance, or health insurance?*

Caller: *Life insurance please.*

System: *Life insurance, okay. First, please tell me your age.*

Caller: *Actually... do you have car insurance for my Ford?*

System: *Sure! What model of Ford do you drive?*

To realize this, a grammar as shown in the following excerpt must be defined in the Input object *Ask for age* and **Grammar Control** must be set to true:

```
<grammar version="1.0" mode="voice" root="ROOT">
  <rule id="ROOT" scope="public">
    <item>
      <ruleref id=”Filler”/>
      car insurance for my
      <ruleref id=”CarMake”/>
      <tag>
         vogrammarcontrol='varCarManufacturer='+$CarMake+';
                          gosub=ConfOutput,CarInsurance;
                          goto=Root'
      </tag>
    <item>
...
```

The string `varCarManufacturer='+$CarMake+'`, which will result in something like `varCarManufacturer=Ford` at call time, tells the server to set the Variable object with reference ID *CarManufacturer* to the value *Ford*. The string `gosub=ConfOutput,CarInsurance` makes the server process the Output object *ConfOutput*, followed by the Module object *CarInsurance*. The string `goto=Root` finally

Copyright © 2001-2011 Voxeo Germany GmbH

instructs the server to jump to the Module object with reference ID *Root*. If this was left out the server would continue with reprocessing the object where the caller left off, which is not what is wanted in this example.

A second example shows how to implement implicit correction. Consider the following dialog:

System: *Prime Car Insurance, always at your service. What's the make and model of the car you drive?*

Caller: *Ford.*

System: *And what model of Volvo do you drive?*

Caller: *No, Ford!*

System: *I'm sorry. So what model of Ford do you drive?*

To realize this, the grammar could set the *vogrammarcontrol* slot to the following value:

```
vogrammarcontrol='varCarManufacturer=Ford;gosub=ApologyOutput;continuation=return'
```

This will make the server re-assign the Variable *CarManufacturer* and reprocess the current object (which is *Ask for Model*), resulting in the desired behavior.

Record utterances specifies whether utterances made in this object are to be recorded (*True*) or not (*False*), given that utterance recording is switched on for this session. *Always* indicates that utterances in this object are always recorded when utterance recording is enabled on the service, regardless of statistical filtering. By default the value is *Default*, indicating that the current value of the corresponding dialog context setting RECORDUTTERANCES is used.

Recording scope defines the scope in which recordings will be made. If set to *Recognition*, only those utterances will be recorded that were successfully recognized (including the activation of hyperlinks). If set to *No Match*, only those utterances will be recorded that triggered a No Match event. If set to *All*, any recording will be stored. If set to *Default*, the current value of the corresponding dialog context setting RECORDINGSCOPE is used. By default the value is *Default*.

Due to the nature of recordings, the feature of recording utterances is only available in the voice and video channel. For more information on this refer to Chapter 6 – *Recording of Utterances* in the *Deployment Guide*.

VoiceObjectsXML Definition

The Input object is represented by the VoiceObjectsXML element `<input>`. It has six attributes and ten children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<input>` element uses the embedded **<**`resultHandling`**>** and **<**`item`**>** elements.

Input

Attributes

- **`autoAdvanceNoInput`**
  Defines whether the dialog should automatically advance to the next object in case of a No Input event. Either *true* or *false*. If not specified, defaults to *false*.
- **`autoAdvanceNoMatch`**
  Defines whether the dialog should automatically advance to the next object in case of a No Match event. Either *true* or *false*. If not specified, defaults to *false*.

Copyright © 2001-2011 Voxeo Germany GmbH

- **maskCallerInput**
  Defines whether the input made by the caller should be masked when writing it to Infostore. This is recommended for sensitive data such as e.g. banking account numbers or PINs.
  One of *true*, *false*, or *complete*. If not specified, defaults to *false*.
- **recordUtterances**
  Defines whether to use utterance recording. One of *true*, *false*, *always*, *default*. If not specified, defaults to *default*.
- **recordingScope**
  Defines the scope for utterance recording. One of *all*, *noMatch*, *recognition*, *default*. If not specified, defaults to *default*.
- **increasedDialogTimeout**
  Defines the increased dialog timeout in seconds. If not specified, the setting from the dialog context is applied at call time.

Children

- **<expression usage="precondition">** or
  <variable usage="precondition"> or
  <collection usage="precondition"> or
  <script usage="precondition">
  Defines the precondition for the Input object.
- **<sequence usage="preprocessing">**
  Defines the preprocessing sequence for the Input object.
- **<output type="initial">**
  Defines the "initial" section of the embedded output.
- **<output type="reprompt">**
  Defines the "event reprompt" section of the embedded output.
- **<grammar>**
  Defines the grammar to be used for accepting caller input.
- **<resultHandling>** [required]
  Defines the slot returns (including related values such as confidence) and their assignment to variables.
- **<eventHandling>**
  Defines the event handling for the Input object.
- **<customNavigation>**
  Defines the custom navigation for the Input object.
- **<tuning>**
  Defines the tuning settings for the Input object.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the Input object.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<input name=”Get car manufacturer”>
  <output type=”initial”>
    <outputItem language=”en-US”>
     <text>Please let us know which car you drive.</text>
    </outputItem>
  </output>
  <output type=”reprompt”>
    <outputItem language=”en-US”>
     <text>We need to know which car you drive. Please say the
     manufacturer, followed by the model.</text>
    </outputItem>
  </output>
  <grammar link=”#Car grammar”/>
  <resultHandling>
    <item alias=”sltManufacturer” object=”#Car manufacturer”/>
    <item alias=”sltModel” object=”#Car model”/>
  </resultHandling>
</input>
```

ResultHandling

Children

- **+<item>**
  Defines the list of result handling items. The attributes `alias` and `object` need to be defined in each item.

Example

```
<resultHandling>
  <item alias=”sltMake” object=”#Car manufacturer”/>
  <item alias=”sltModel” object=”#Car model”/>
</resultHandling>
```

Item

Attributes

- **object**
  Defines a reference to an object.
- **alias**
  Defines the slot name. For TTG grammars, this is *sltTTG* by default.
- **removeSpaces**
  Defines whether space characters will be removed from slot return values before being assigned to the Variable or Layer object. Either *true* or *false*. If not specified, defaults to *false*.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<item alias=”premium” object=”#Annual premium”/>


<item alias=”phone” object=”#Phone” removeSpaces=”true”/>
```

Object Interoperability

The following table contains all object types that can reference an Input object:

Icon	Object Name	Use Case Example
	Module	An Input object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, an Input object is used within a sequence in a dialog flow.
	Menu	An Input object can be used as the destination object within a Menu item.
	Confirmation	An Input object is frequently used as the destination object within a Correction item inside a Confirmation object.
	If	An Input object can be used in either the THEN item or the ELSE item of an If object.
	Case	An Input object can be used in any WHEN item and the ELSE item of a Case object.
	Loop	An Input object can be used within the embedded sequence of a Loop object.
	Goto	An Input object can be referenced via a Goto object.
	Hyperlink	An Input object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of an Input object should indicate which type of input is requested from the caller. The short description should contain a brief explanation of the slots that are filled by the input. The table below lists three examples:

Name	Description
 Ask for phone number	Requests a phone number from the caller and places it in the variable *Phone Number*.
 Confirm transaction	Replays the transaction to be performed and asks for an explicit yes/no confirmation. The response is stored in the variable *Yes/No Answer* as *yes* or *no*.
 Ask for car make and	Multi-slot input asking for both maker and model of

Copyright © 2001-2011 Voxeo Germany GmbH

Name	Description
model	a car. Results are stored in *Car make* and *Car model*. It is possible to say only the make, in which case *Car model* will be empty.

Copyright © 2001-2011 Voxeo Germany GmbH

Layer

Overview

The Layer object represents one of the core features of VoiceObjects. It allows the designer of an application to separate the application logic from the way the dialog is presented to the caller. Layers can influence aspects such as language, channel or persona, but they can also change the application behavior itself. For instance, a menu can have additional choices for gold customers depending on specific profile data available from a back-end system, or certain application features can be disabled outside normal business hours.

The Layer object is the place to specify a custom layer of an application. The Layer object definition always lists all states the corresponding layer can be in.

VoiceObjects also provides standard layers, the so-called *system layers*, which are language, channel, and input mode, plus the object-specific occurrence level. They are frequently required in applications and need not be implemented manually. They are present in many objects and can readily be used within the corresponding object editors. All other layers, which are rather application-specific such as persona, caller preferences, service level etc., are each defined within a single Layer object.

Infostore, the logging component of VoiceObjects Server, automatically keeps track of layer states for each call, if desired, to provide the data for reports on personalization in VoiceObjects Analyzer.

VoiceObjects distinguishes between manual layers and automatic layers.

Manual layers

Manual layers need to be switched manually during call time, typically activated by some command or choice made by the caller. The switch is performed by using the system function LAYER(layer, [state]) within an Expression object, which allows you to either retrieve the current state of a layer (leaving argument *state* empty), or to explicitly set it to a new state (providing argument *state* as the state to switch to). Manual layers can only be in one state at a time. The most frequent example of a manual layer is *Persona*. The number of voices you want to have in your application corresponds to the number of states of this layer, one of which is declared as the default (= initial) state. By using a Hyperlink object with a corresponding Expression object as its destination, the layer can be switched to a different state at call time, so that all subsequent prompts are spoken by the corresponding persona.

Automatic layers

Automatic layers change their states automatically, according to the status of the application during call time. Examples would be *Caller Profile* depending on back-end data, or *TimeOfDay* depending on the current time of day. Automatic layers, as opposed to manual layers, may have more than one active state. The state definition is typically done by analyzing the values of variables or expressions, which usually change during a dialog, e.g. due to back-end access or caller inputs.

For more information on the concept and use of layers, see Chapter 7 – *How to Use Layers* in the *Design Guide*.

The Object Definition below covers the configuration of the Layer object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Layer object belongs to the object category Layers.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Flow Scenario

The following dialog flow example demonstrates how the caller can switch the persona using a hyperlink command. The Menu object holds two initial prompts that are associated with the manual layer *Persona* and corresponding states *male* and *female*. The layer is initially in state *male*, so this prompt is played first. After switching the layer to state *female*, the corresponding prompt is played. The selected prompt for playback is indicated by bold type:

Object – Caller		Dialog Flow
	**male**	[male voice:] ***This is the main menu. Please make your choice: news, sports, or entertainment.***
	female	[female voice:] *You are in the main menu. Please choose between news, sports, and entertainment.*
 Caller		*Switch to female voice.*
	male	[male voice:] *This is the main menu. Please make your choice: news, sports, or entertainment.*
	female	[female voice:] ***You are in the main menu. Please choose between news, sports, and entertainment.***

The following example shows the effect of a layer *TimeOfDay* that affects the selection of a welcome prompt defined in an Output object. The layer has the four states *morning*, *afternoon*, *evening*, and *night*. Assume the caller calls in at 2pm. The correct prompt is played, indicated by bold type. The second Output object is not associated with a layer, so there is only one prompt defined. The final output in this example would thus be "*Hello! Welcome to Prime Insurance*".

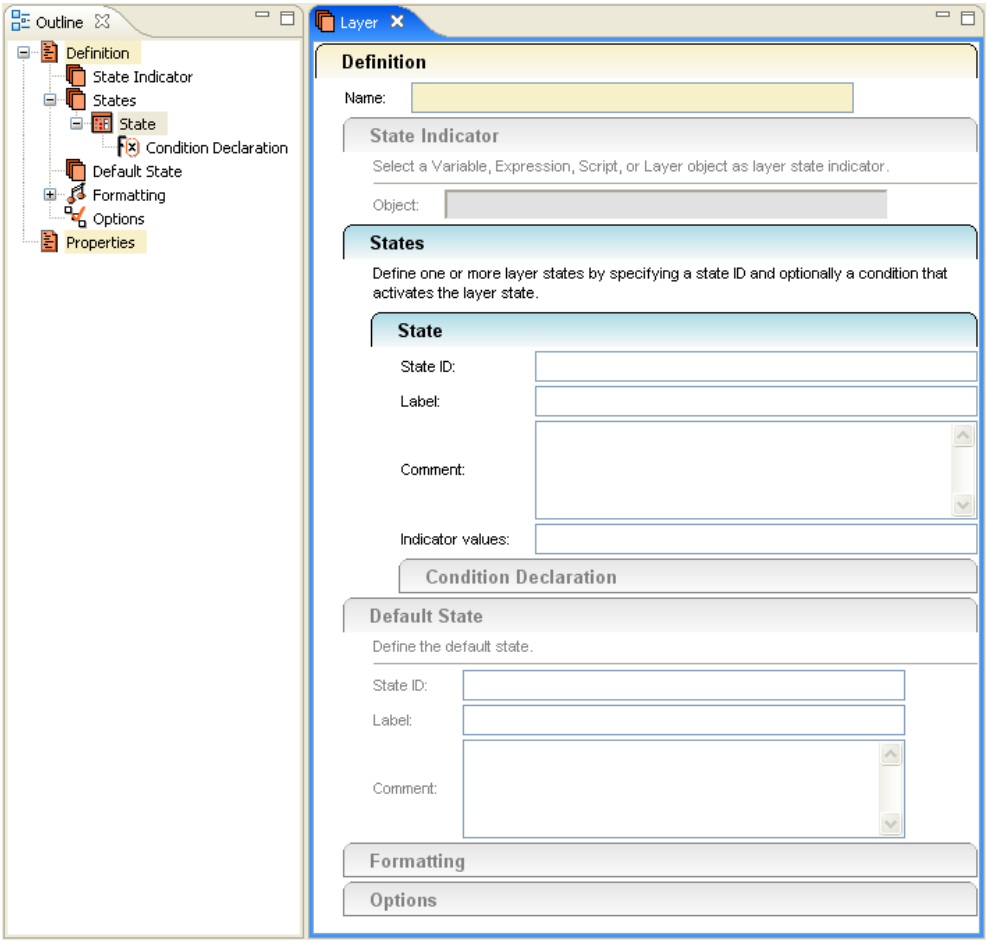
Object – Caller		Dialog Flow
	morning	*Good morning!*
	afternoon	***Hello!***
	evening	*Good evening!*
	night	*Hello!*
		***Welcome to Prime Insurance***.

Copyright © 2001-2011 Voxeo Germany GmbH

Object Definition

The **Definition** of the Layer object provides the following sections:

- State Indicator
  To reference a Variable, Expression, Script, or another Layer object to compare values against (only for automatic layers).
- States
  To list all available states for this layer.
- Default State
  To define a default state.
- Formatting
  To define formatting instructions for playback within an Output object.
- Options
  To enable or disable the logging of layer states.



For information regarding the **Properties** section, refer to *Properties* in this *Object Reference*.

Manual and automatic layers differ in the way they are defined in the **Layer** editor:

Copyright © 2001-2011 Voxeo Germany GmbH

Define manual layers

Manual layers are defined by specifying each individual state by means of their respective state ID and label, plus a default state. The obligatory default state is a full state definition on its own. Therefore it is not possible to list a state within the **States** section and at the same time define it in the **Default State** section, using the same state ID. The **State Indicator** is always left empty for manual layers.

Define automatic layers

Automatic layers can be defined in two different modes:

1. Provide a state indicator object and define the states by comparing the content of this object to some given values (static or dynamic). In this case, you do not define a condition declaration for any **State** definition within the **States** section. Use this mode if the states all depend on the values of one specific object of your project, typically a Variable or Expression object.
   In this mode, a layer state is active if the value of the state indicator object equals one of the comparison values provided for this state. Multiple states may be active at the same time.
2. Specify the states by explicit condition declarations for all **State** definitions within the **States** section. In this case, you must not define a state indicator object. Use this mode if the states are rather unrelated to each other, and they all depend on distinct conditions.
   In this mode, a layer state is active if its condition evaluates to true. Multiple states may be active at the same time.

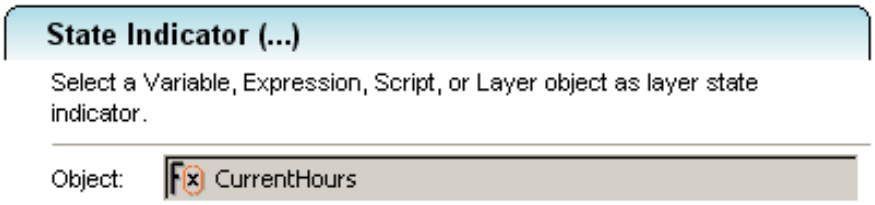
In both modes of automatic layers, the **Default State** section defines a state that is active if none of the other states are active. For automatic layers, the definition of a default state is optional.



Caution: Mixing both modes for automatic layers, e.g. defining a state indicator object and also specifying condition declarations for some states is not possible and will lead to an internal error on VoiceObjects Server at call time.

State Indicator

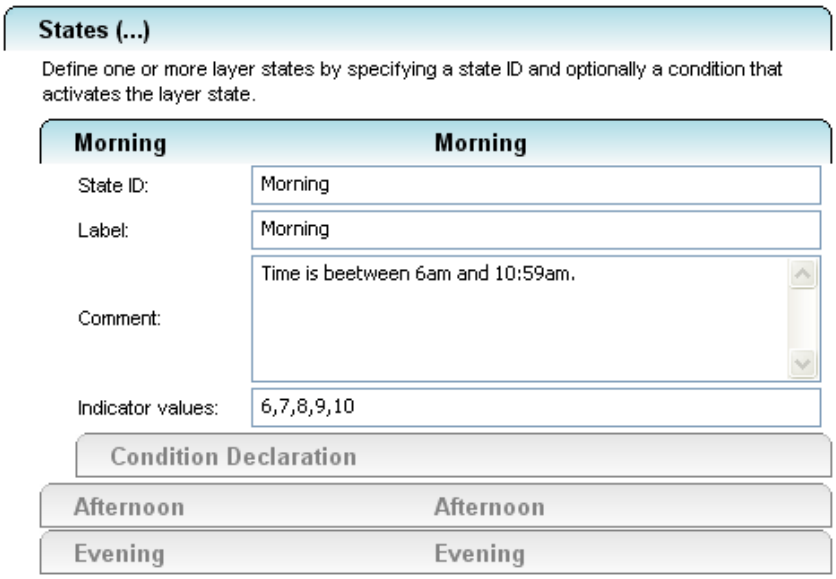
The **State Indicator** only needs to be defined for automatic layers. If you make use of a state indicator (see Define automatic layers for a description of the two different modes of automatic layers), link a Variable, Expression, Script, or some other Layer object as a state indicator object. The content of this object is compared to the indicator values specified in the **States** and **Default State** sections. If the content is identical to one of the values defined as indicator values, then this state is an active state of the layer. If none of the states defined in the **States** section is active, the default state (if defined) is the only active state of this layer.



Copyright © 2001-2011 Voxeo Germany GmbH

States

The **States** section lists all states the layer can be in, except for the default state, which is an additional state and defined in the **Default State** section.



In each **State** the following properties can be specified:

Property	Description
State ID	Mandatory parameter that identifies a state within a layer. It must be unique across the project, i.e. there cannot be two states – independent of the Layer object – with the same state ID in your project. State IDs must not contain blanks or other special characters except hyphens or underscores.
Label	Used for display purposes of layer states outside the **Layer** editor. In all editors that allow you to associate layer conditions, the label is shown instead of the state ID. If you do not specify a label, the state ID is copied into this field when saving the Layer object. The label must be unique across the project, i.e. there cannot be two states – independent of the Layer object – with the same label in your project. Labels must not contain the characters '=' or '!=' to prevent confusion in layer state conditions.
Description	Text area that allows you to provide a description of this state within the layer, for documentation purposes (optional).
Indicator Value	When defining an automatic layer via a state indicator (first mode, as described in Define automatic layers), use this field to define a value or a comma-separated list of values to compare the content of the state indicator object to. If the content of this object is equal to one of the indicator values, then this state is an active state of the layer. You can also link a Variable, Expression, Collection, Script, or Layer object to dynamically define the indicator values at call time.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Condition Declaration	When defining an automatic layer using condition declarations (second mode, as described in Define automatic layers), use this section to specify the condition that must be met for the corresponding state to be active. Either define an embedded condition by selecting an appropriate operator from the function drop-down list and specifying the required arguments, or link a Variable, Expression, Collection, Script, or Layer object that returns a Boolean value. For more information on the use of functions, see the Expression object in this *Object Reference*.

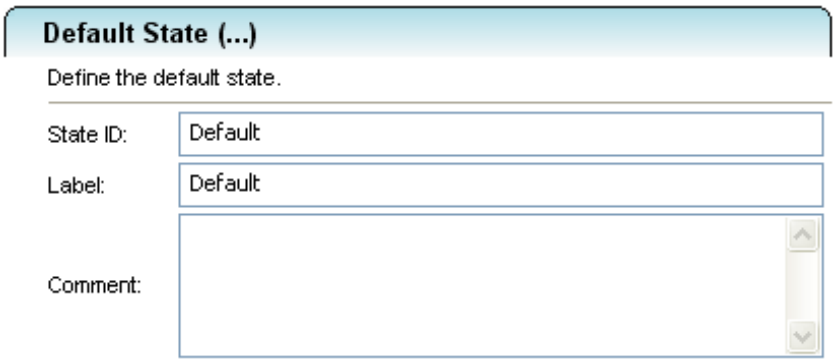
Manual layers are defined by specifying a state ID, a label, and optionally a description. Automatic layers additionally require indicator values or a condition declaration, depending on the mode used (see Define automatic layers).

Note: Be careful when setting the labels and state IDs of layer states. Avoid swapping or renaming layer states once a Layer object has been created, as this might lead to problems in the object structure when saving the object.

Default State

The **Default State** definition can be interpreted in two different ways, depending on the type of layer:

- For manual layers, the default state is the initial state the layer is in when the dialog starts. The state is defined the same way as in the **States** section, by providing a state ID, a label and optionally a description.
- For automatic layers, the default state is the active state in case that all other states defined in the **States** section are not active, i.e. no other state condition evaluates to true in the corresponding dialog step at call time. Because of this, there is no need to specify any conditions in the default state. The default state for an automatic layer is therefore defined the same way as for manual layers, by providing a state ID, a label and optionally a description.

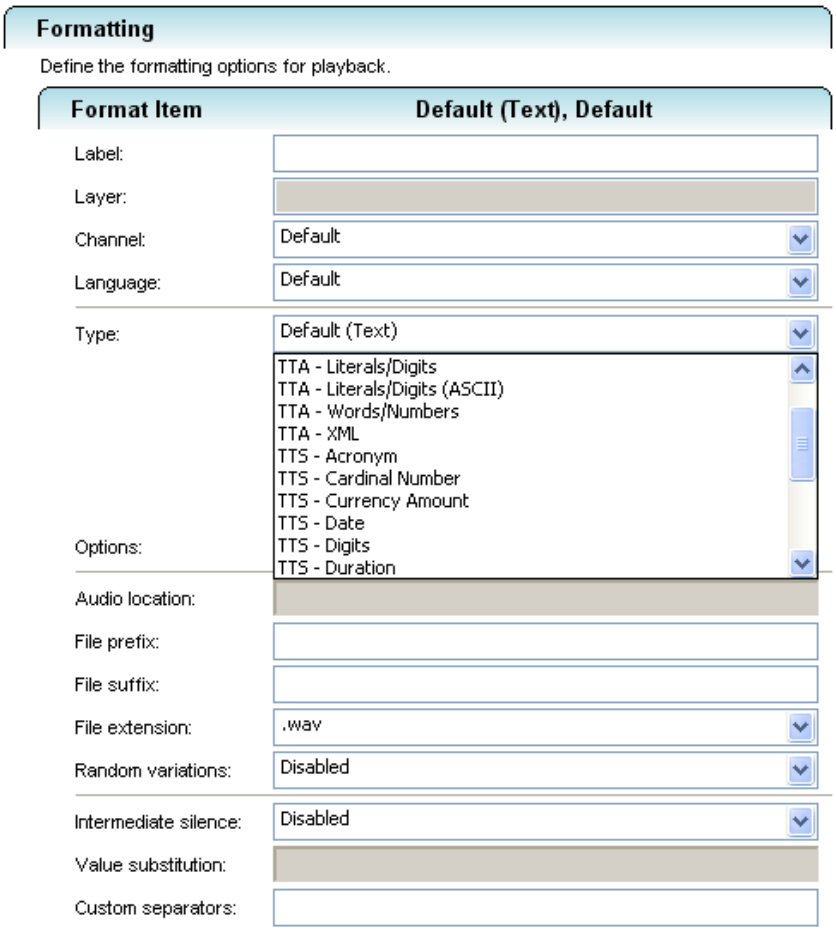


Note: The default state is a separate state definition. It is not possible to define a state in the **States** section and then repeat this state in the **Default State** section, since each state ID must be unique across all layers within a project.

Copyright © 2001-2011 Voxeo Germany GmbH

Formatting

The **Formatting** section defines how the active state of a Layer object is presented to the caller when it is used within an Output object.



Two different categories of formatting types are offered, which are only relevant for the voice and video channel:

- **Text-To-Audio (TTA)**
  The object content is played back to the caller through the concatenation of prerecorded audio files.
- **Text-To-Speech (TTS)**
  The object content is played back to the caller through text-to-speech synthesis provided by the media platform.

The formatting instructions can be defined as an embedded definition as part of the Layer object, or can be linked to an autonomous Format object.

To define formatting types for the text and Web channel, use the Formatting bus as described in Appendix B – *How to Use the Formatting Bus* in the *Administration Guide*.

For further details on formatting capabilities refer to the Format object in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Options

The **Options** section specifies if logging of layer states to Infostore is enabled or disabled.



If the option **Enable Layer State logging** is selected, Infostore will log the state the layer was in when the call was finished. More precisely, it is the state the layer was set to the last time it was used in a call. This means, e.g., evaluated as a precondition or changed through the LAYER function. Automatic layers can be in more than one state, in which case a set of states will be stored.

General information about Layer objects, such as names, layer states, creation and modification timestamps etc., is additionally logged to Infostore with every redeploy or restore of an application. Thus, if you make bigger changes to a Layer object such as deleting states, its information will simply be updated so that old Infostore data might not be usable anymore. In that case, consider creating a new Layer object instead of modifying an existing one.

See Chapter 3 – *The Physical Data Model* in the *Infostore Guide* for detailed information on the table structure, and Chapter 3 – *Report Library* in the *Analyzer Guide* for information on available reports on Layer State logging.

Usage Examples

The following two examples demonstrate how to build the manual layer *Persona* and the automatic layer *TimeOfDay*, as introduced in the Dialog Flow Scenario. All necessary definitions in the **Layer** editor are described, followed by the steps required to use the layers within other objects.

Manual layer Persona

This layer represents two personas of your application, *male* and *female*.

1. Create a new Layer object.
2. Enter *Persona* as the name of the Layer object.
3. In the **States** section, define the first **State** by entering *male* both as the state ID and as the label.
4. Define the second layer state as the default state by entering *female* as state ID and label in the **Default State** section.
5. Save this Layer object.

You can use this Layer object to provide different prompts, e.g. different prerecorded audio files, in your application corresponding to the selected persona. There are basically two ways to do this, as described in Chapter 7 – *How to Use Layers* in the *Design Guide*. Either use the Layer object inside a Resource Locator object to dynamically change the path to the audio files during call time, or explicitly associate the layer with different Output items within an Output object. For the layer *Persona*, the first approach is more adequate, since different personas typically only result in different prompts to be played while the dialog structure stays the same. See Resource Locator in this *Object Reference* for a description of how to embed a Layer object into the definition of a Resource Locator object.

Copyright © 2001-2011 Voxeo Germany GmbH

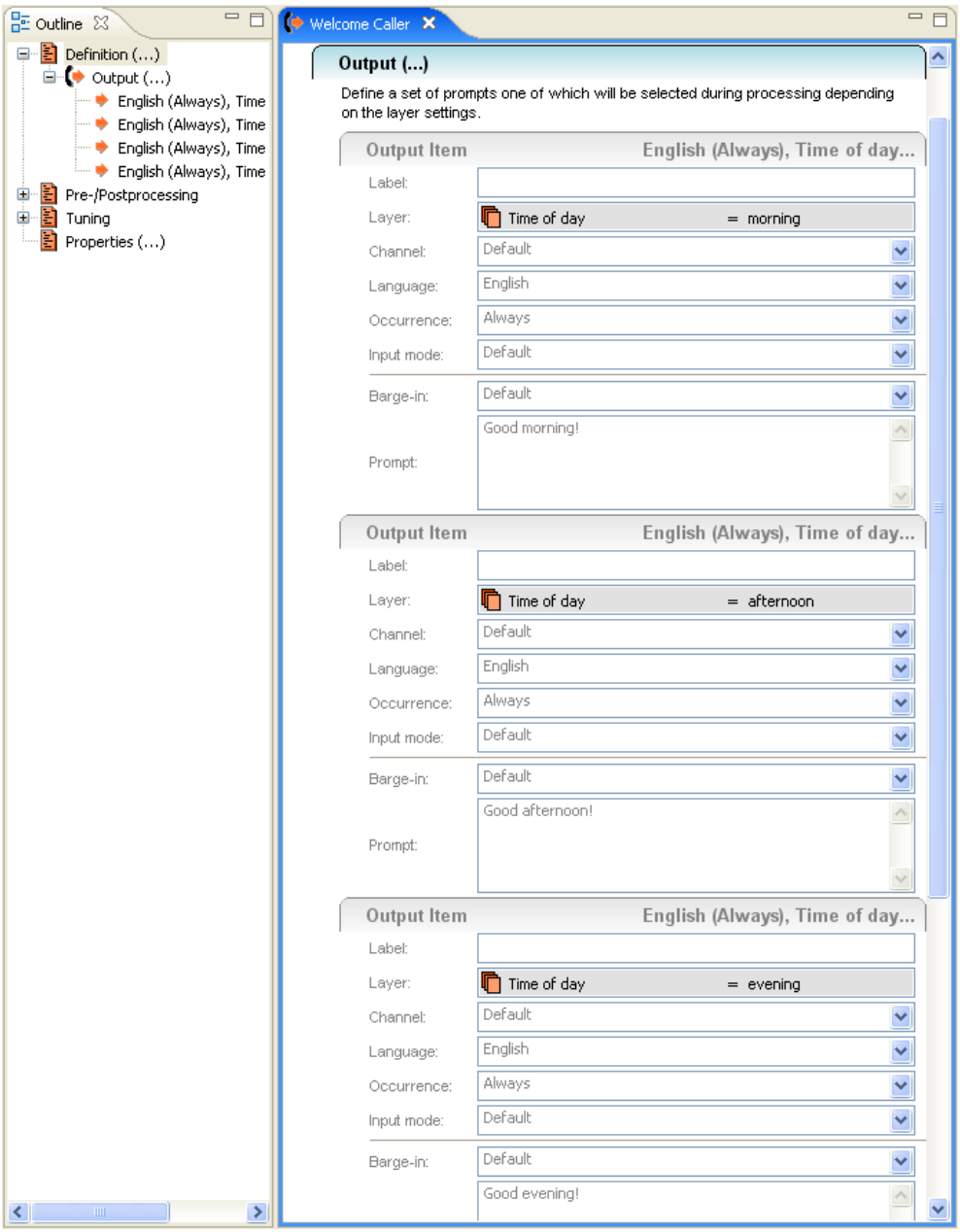
Automatic Layer TimeOfDay

This layer represents different times of day, with states *morning*, *afternoon*, *evening*, and *night*. It is implemented in the first mode (see Define automatic layers), using a state indicator object. In addition to the Layer object itself, an Expression object is necessary to retrieve the current time.

1. Create a new Layer object.
2. Enter *TimeOfDay* as the name of the Layer object.
3. In the **State Indicator** section, insert a new Expression object into the **Object** field.
4. Enter *CurrentHours* as the name of the Expression object.
5. From the **Function** drop-down list, select *NOW*.
6. Enter *HH* as first argument to always retrieve the current time as full hours (0-23).
7. Save and close this Expression object to link it to the **State Indicator** section of the Layer object.
8. In the **States** section, define the first **State** by entering *morning* both as the state ID and the label. Enter *8, 9, 10, 11* as the indicator values for this layer state. Enter *From 8am to 11:59am* as the description of this state.
9. Define a second **State** by entering *afternoon* as state ID and label. Enter *12, 13, 14, 15, 16, 17* as the indicator values for this layer state. Enter *From noon to 5:59pm* as the description of this state.
10. Define a third **State** by entering *evening* as state ID and label. Enter *18, 19, 20, 21* as the indicator values for this layer state. Enter *From 6pm to 9:59pm* as the description of this state.
11. Define a fourth layer state as the default state by entering *night* as state ID and label in the **Default state** section. Enter *From 10pm to 7:59am* as the description of this layer state.
12. Save this Layer object.

This Layer object is typically used to play a time-dependent greeting at the beginning of a dialog. The following screenshot taken from an Output object definition shows how to use different layer conditions of the *TimeOfDay* layer to distinguish between several Output items.

Copyright © 2001-2011 Voxeo Germany GmbH



VoiceObjectsXML Definition

The Layer object is represented by the VoiceObjectsXML element `<layer>`. It has two attributes and three groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<layer>` element uses the embedded `<layerState>` and `<defaultLayerState>` elements.

Layer

Attributes

The standard attribute `name` is required for Layer objects.

Copyright © 2001-2011 Voxeo Germany GmbH

- **`stateIndicator`**
  Defines the layer state comparison value. Must be a reference to an Expression, Script, or Variable object.
- **`layerStateLogging`**
  Defines if Layer State logging is to be enabled or not. Set it to *true* (the default) to enable it, and to *false* otherwise.

Children

- +**`<layerState>`**
  Defines the set of states for the Layer object.
- **`<defaultLayerState>`**
  Defines the (optional) default state for the Layer object.
- **`<format>`**
  Defines the output formatting for the layer return value.

Example

```
<layer name=”DayStatus” stateIndicator=”#Today”>
  <layerState stateID=”weekday”
values=”monday,tuesday,wednesday,thursday,friday”/>
  <layerState stateID=”weekend” values=”saturday,sunday”/>
</layer>

<layer name=”Persona”>
  <layerState stateID=”Male”/>
  <defaultLayerState stateID=”Female”/>
</layer>
```

LayerState

Attributes

- **`stateID`** [required]
  Defines the ID of the layer state. Must be unique across all Layer objects. State IDs must not contain blanks or special characters other than underscore.
- **`label`**
  A text string providing a name for the layer state. If not specified, the state ID is used as label.
- **`shortDescription`**
  A text string providing a description for the layer state.
- **`values`**
  Defines either a comma-separated list of constant values, or a reference to an Expression, Script, or Variable object. Either this attribute is defined, or the child described below is defined, but not both.

Copyright © 2001-2011 Voxeo Germany GmbH

Children

- **<expression>** or
  <variable> or
  <collection> or
  <script>
  Defines an embedded object as condition for the layer state.

Example

```
<layerState stateID=”premium”>
  <expression function=”greater”>
    <argument><variable link=”#Balance”/></argument>
    <argument><value>10000</value></argument>
  </expression>
</layerState>
```

DefaultLayerState

Attributes

- **stateID** [required]
  Defines the ID of the default layer state. Must be unique across all Layer objects. State IDs must not contain blanks or special characters other than underscore.
- **label**
  A text string providing a name for the default layer state. If not specified, the state ID is used as label.
- **shortDescription**
  A text string providing a description for the default layer state.

Example

```
<defaultLayerState stateID=”premium”/>
```

Object Interoperability

Layer objects can be referenced in all objects that provide layers and/or have a precondition setting. They can also be used within an Output, Log or Database object to play back their state to the caller or log it to a database or file. Furthermore, layer conditions can appear as arguments in Expression objects, as well as values in OSDM objects.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The description of a Layer object should mention the type of the layer (manual or automatic) and also list all possible states of this layer. In case of automatic layers, describe the conditions that need to be met for the individual states.

The table below shows three examples:

Copyright © 2001-2011 Voxeo Germany GmbH

Name	Description
Persona	Manual layer that covers all personas available in the application. Possible states are *robert*, *marc*, *lisa*, and *susan*.
TimeOfDay	Automatic layer that covers four times of the day: *morning* (from 7am to 11am), *day* (from 12pm to 5pm), *evening* (from 6pm to 10pm), and *night* (all other hours).
Customer Status	Manual layer that represents the status of the caller. Possible states are *standard*, *premium*, or *gold*.

Copyright © 2001-2011 Voxeo Germany GmbH

List

Overview

The List object allows callers to navigate through a list of items and perform various actions on an individual item, e.g. reading, deleting, forwarding a message in an e-mail reader or selecting an item from a shopping list. The underlying data structure for the List object is represented by a Collection object and can either be a one-dimensional list (consisting of any number of rows) that only allows vertical navigation (using commands like *next item*, *go back*, *repeat please* etc.), or a two-dimensional table (consisting of any number of rows plus a predefined set of columns) that additionally provides horizontal navigation to jump to a column directly. In the latter case, the designer must explicitly specify which columns of the underlying table are to be read out to the caller. The designer can choose to present every column of the table, or only a subset thereof. It is also possible to present the data of more than one column together as one presentation unit. For this, the List object introduces the notion of *column group* to define the presentation units for each List item. A *List item* is equivalent to a row in the underlying collection. The List object presents its information item by item. This is also the case in the text and Web channel; the list data will *not* be displayed all at once.

The content of the list can either be defined statically at development time or retrieved dynamically during the dialog, e.g. from an external data source like a database system or a simple file. The List object also provides an optional selection cart (e.g. a virtual shopping cart in a shopping application) that stores selected items for later re-use.

The Object Definition below covers the configuration of the List object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The List object belongs to the object category Components.

Dialog Flow Scenario

The following dialog flow example shows how the caller navigates through a stock portfolio that offers quotes, news, and the number of shares the caller owns of each company in the list.

The table that serves as the content for the list looks as follows:

Grammar	Name	Quote_today	Quote_ yesterday	News	Portfolio
ibm	IBM	17.5	16.3	None.	56
apple, apple computers	Apple	23.8	21.4	Apple will release their new operating system tomorrow.	127
dell	Dell	56.1	58.0	Dell’s newly introduced laptop line is flying off the shelves.	83

Copyright © 2001-2011 Voxeo Germany GmbH

Given this table, it is up to the designer to decide which columns to put into column groups. The dialog flow example below shows three column groups: The first one presents today's and yesterday's quotes together as one prompt; the next one reads out the news, whereas the last one presents the number of shares.

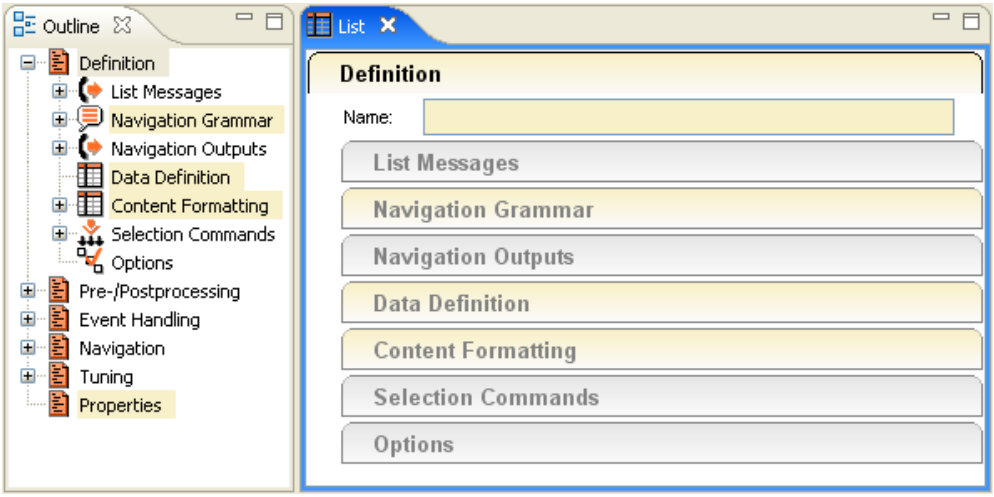
Object – Caller	Dialog Flow
	*Here you can check quotes and news from companies of your stock portfolio. Please use the commands **next**, **back**, and **repeat** to navigate through the list. You can either simply listen to all information or say **quotes**, **news**, or **shares** to jump to these items directly. You can also navigate to a particular company by saying its name. To exit the table say **stop**.*
	*IBM. Their quote yesterday was 16.3, today it's 17.5.* (auto-advancing to next column group) <Earcon> *News: None.* (auto-advancing) <Earcon> *You own 56 shares of IBM.*
	(Does not say anything)
	(Auto-advancing to next row) *Apple. Today's quote is 23.8…*
	(Barges in) *News*
	*News: Apple will release their new operating system tomorrow.* (auto-advancing) <Earcon> *You hold 127 shares of Apple.*
	*Back*
	(Moving to the previous row) *IBM. Their quote yesterday was 16.3, today it's 17.5.* (auto-advancing) <Earcon> *News: None.* (auto-advancing) <Earcon> *You own…*
	(Barges in) *Back*
	(Trying to move to the previous row) *This was the first entry in your list. You cannot go back here. IBM. Their quote…*
	(Barges in) *Dell*
	*Dell. The quote today is 56.1, yesterday's close was 58.* (auto-advancing) <Earcon> *News: Dell's newly introduced laptop line is flying off the shelves.* (auto-advancing) <Earcon> *You own 83 shares of Dell.*
	*Stop*

Copyright © 2001-2011 Voxeo Germany GmbH

Object Definition

The **Definition** of the List object provides the following sections:

- List Messages
  To specify various kinds of optional messages related to the list as a whole.
- Navigation Grammar
  To provide grammars for vertical and horizontal navigation through the list.
- Navigation Outputs
  To define optional outputs to be presented after the caller has activated a navigation command; as well as presentation outputs for the commands, which are mandatory for the text and Web channel.
- Data Definition
  To specify the content of the list and other information related to it.
- Content Formatting
  To define sets of table columns to be played back as one presentation unit.
- Selection Commands
  To specify the commands to be performed on a particular List item.
- Options
  To provide processing options.

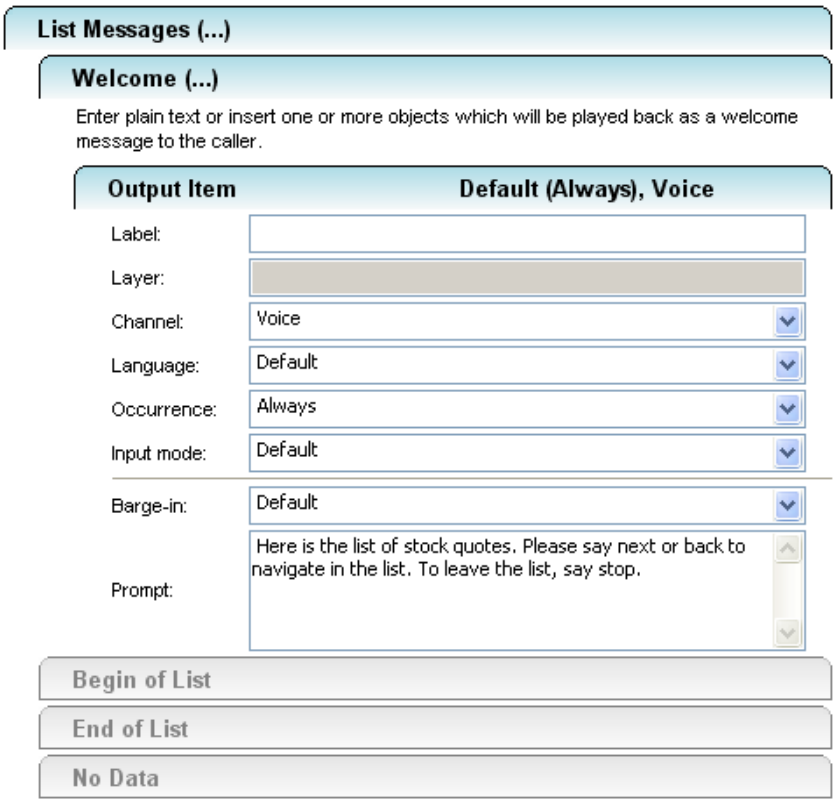


For further details regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Navigation*, *Tuning*, and *Properties* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

List Messages

The **List Messages** section contains four subsections to specify the following optional messages through embedded Output objects.



- **Welcome** is presented as the caller enters the list.
- **Begin of List** is presented if the caller tries to navigate backward beyond the first item of the list.
- **End of List** is presented when the end of the list is reached, either through auto-advance or caller navigation.
- **No Data** is presented if the Collection object defining the list does not contain any data.

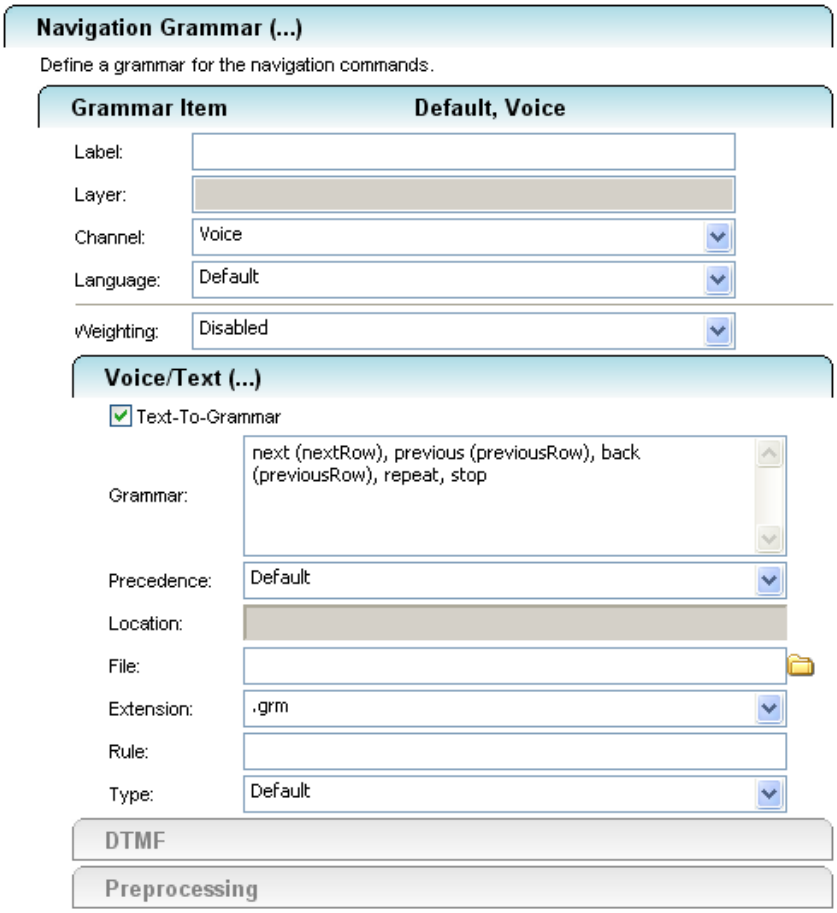
Instead of configuring these outputs as embedded definitions within the List object, existing Output objects can be referenced by over-linking the embedded definitions.

For further details, refer to the Output object.

Navigation Grammar

In the **Navigation Grammar** the possible input for vertical navigation among rows and horizontal navigation among column groups is specified. This covers the generic commands for relative navigation only, such as *back*, *previous row*, *next item* etc. Grammars for direct row access are defined within the data collection (see Data Definition), and grammars for direct column group access are defined in the **Content Formatting** section.

Copyright © 2001-2011 Voxeo Germany GmbH



You can define appropriate grammars for the following commands:

- **Start** (*sltNavigation = start*)
  Navigates to the first column group of the first row in the list. If the caller already is at this position, this command has the same effect as *Repeat*.
- **End** (*sltNavigation = end*)
  Navigates to the first column group of the last row in the list. If the caller already is at this position, this command has the same effect as *Repeat*.
- **Previous** (*sltNavigation = previous*)
  Moves back one step in the navigation history. The navigation history keeps track of all column groups passed. If there is no previous step in the history, the caller will move up one row. In this case, the setting of the option **Maintain column group during navigation** (see Options) indicates if the first column group is then presented (if check box is clear), or the one the caller was in when activating this navigation command (if check box is selected). If the current row is the first one, the **Begin of List** message will be played (if defined).
- **Next** (*sltNavigation = next*)
  Moves forward one step in the navigation history (as an undo to a *Previous* command). If there is no next step in the history, the caller will move to the next row. The setting of the option **Maintain column group during navigation** (see Options) indicates if the first column group is then presented (if check box is clear), or the one the caller was in when activating this navigation command (if check box is selected). If there is no next row, the **End of List** message will be played (if defined).

Copyright © 2001-2011 Voxeo Germany GmbH

- **First Column Group** (*sltNavigation = firstCG*)
  Navigates to the first column group of the current row. If the caller already is in the first group, this command has the same effect as *Repeat*.
- **Last Column Group** (*sltNavigation = lastCG*)
  Navigates to the last column group of the current row. If the caller already is in the last group, this command has the same effect as *Repeat*.
- **Next Column Group** (*sltNavigation = nextCG*)
  Navigates to the next column group of the current row. If the caller is in the last group, the application jumps to the first column group of the current row.
- **Previous Column Group** (*sltNavigation = previousCG*)
  Navigates to the previous column group of the current row. If the caller is in the first group, the application jumps to the last column group of the current row.
- **First Row** (*sltNavigation = firstRow*)
  Navigates to the first row of the list. If the caller already is in the first row, this command has the same effect as *Repeat*. The setting of the option **Maintain column group during navigation** (see Options) indicates if the first column group is presented (if check box is clear), or the one the caller was in when activating this navigation command (if check box is selected).
- **Last Row** (*sltNavigation = lastRow*)
  Navigates to the last row of the list. If the caller already is in the last row, this command has the same effect as *Repeat*. The setting of the option **Maintain column group during navigation** (see Options) indicates if the first column group is presented (if check box is clear), or the one the caller was in when activating this navigation command (if check box is selected).
- **Next Row** (*sltNavigation = nextRow*)
  Navigates to the next row of the list. If the caller is in the last row, the **End of List** message is played (if defined). The setting of the option **Maintain column group during navigation** (see Options) indicates if the first column group is presented (if check box is clear), or the one the caller was in when activating this navigation command (if check box is selected).
- **Previous Row** (*sltNavigation = previousRow*)
  Navigates to the previous row of the list. If the caller is in the first row, the **Begin of List** message is played (if defined). The setting of the option **Maintain column group during navigation** (see Options) indicates if the first column group is presented (if check box is clear), or the one the caller was in when activating this navigation command (if check box is selected).
- **Repeat** (*sltNavigation = repeat*)
  Plays the current column group again.
- **Stop** (*sltNavigation = stop*)
  Exits the List object without assigning a value to the selection cart (see Data Definition for more details on the selection cart). Note that the selection cart is not reset, but instead retains whatever value it has when the caller exits the List object.

The navigation grammar is required in all steps of the List object, including the **Welcome Message**, and needs to fill the slot *sltNavigation* with the exact values indicated above. Neither the name of the slot nor the values may be different (note, though, that it is possible to define a different slot name through a media platform driver if required). Not all commands need to be defined. It is the designer's decision to offer all or a subset of the available commands to the caller.

An example grammar in TTG format for the voice and video channel might look as follows:

Copyright © 2001-2011 Voxeo Germany GmbH

next (nextRow), previous (previousRow), back (previousRow), repeat, stop

This simple grammar defines commands for vertical navigation (*next*, and the synonyms *previous* and *back*) plus commands to repeat the current column group (*repeat*) and to exit the whole list (*stop*).

In the textual channels, the navigation grammar can only be defined using TTG. In the text channel, the grammar tokens will be displayed as the access keys for the navigation commands. An example grammar might look like this:

4 (previousRow), 6 (nextRow), 9 (stop)

This would assign the keys 4, 6, and 9 to the commands *previous row*, *next row*, and *stop*, respectively. How the commands are presented is defined in the **Navigation Outputs** section.

In the Web channel, grammars are conceptually not required, as they will not be offered to the caller as such. Instead, the presentation outputs of the commands will be displayed as clickable hyperlinks. Nevertheless, the navigation grammar requires naming the slot values of the commands that should be active in the list. Therefore, define the slot values and use the question mark "?" as a placeholder for the grammar tokens:

? (previousRow), ? (nextRow), ? (stop)

If you build an application for both the text and the Web channel, you can use the same grammar definition for both, by setting **Channel** to *Text/Web*.

Tip: Offering all available commands can be more confusing to the caller than helpful, although it would certainly mean most flexibility for frequent callers. The commands *previous* and *next*, which operate on the navigation history, can get confusing when two-dimensional tables are used. The recommended commands to use are *nextRow*, *previousRow*, *repeat*, and *stop*, as these cover the most important navigation options necessary in both list and table navigation.

Note: If there is standard navigation defined for the Module object that contains the List object, be aware that there might be overlaps in the active grammars while being in the list. Grammars defined in the List object have precedence over the standard navigation grammars. So, for example, when the same utterance can trigger the standard navigation command *backward* as well as the list command *previous*, the list command is processed. The behavior of the standard navigation commands *backward*, *forward*, *skip*, and *repeat* is slightly different from that of the list commands *previous*, *next*, and *repeat*. *Backward* is the same as *previous*, but, if on the first List item, *backward* jumps out of the List object, whereas *previous* plays the *Begin of List* prompt (if defined). *Backward* can also undo a selection command, whereas *previous* cannot. *Forward* is the same as *next*, but if the navigation history is empty, *forward* merely repeats the current List item, whereas *next* moves on to the next row. *Forward* also undoes a *backward* command, i.e. after undoing a selection command through *backward*, you can reprocess it via *forward*. *Skip* leaves the List object and proceeds with the next object in the dialog flow. *Repeat* has the same effect for both standard and list navigation.

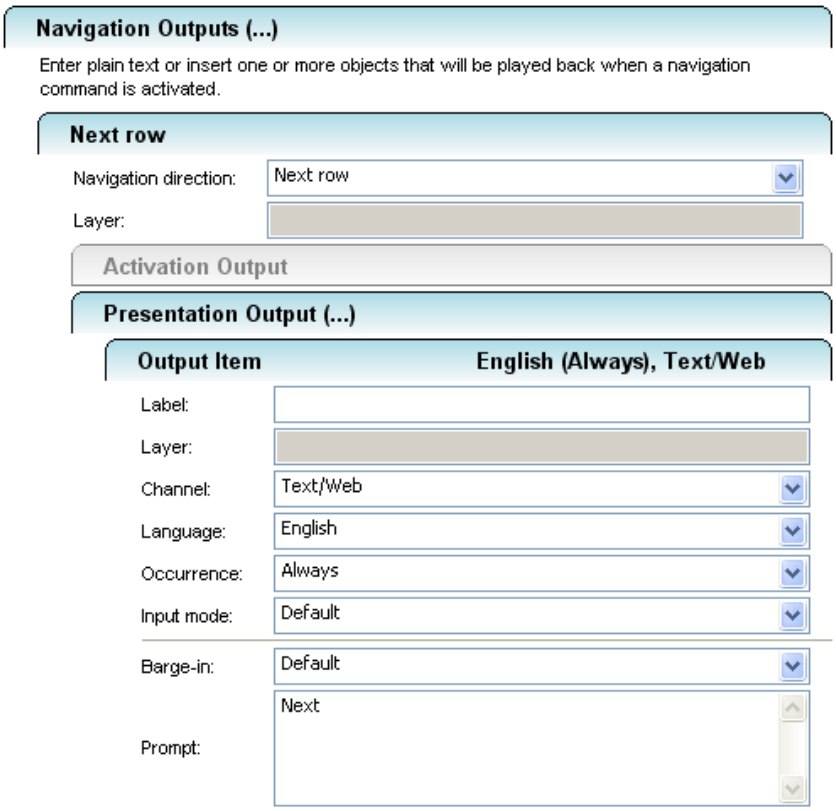
Caution: Some media platforms running services in the voice or video channel do not support slot assignments in DTMF grammars. These platforms always return the DTMF keys pressed, but no semantic interpretation. If you use TTG for DTMF grammars, this will be handled by the server, i.e. in this case slot assignments will work as usual. If you use external, or inline but non-TTG grammars, you need to be aware of this restriction. See Appendix A – *Media Platform Drivers* in the *Deployment Guide* to find out how your media platform handles this.

Copyright © 2001-2011 Voxeo Germany GmbH

Navigation Outputs

In the **Navigation Outputs** section, two kinds of outputs can be defined for all the navigation commands available (plus *auto-advance*).

- **Activation**
  This output is presented after a navigation command has been activated by the caller or after the application moves on to the next row when using *auto-advance*. It is only supported in the voice and video channel, and will be ignored in the textual channels.
- **Presentation**
  This output defines how the navigation command is to be presented to the caller. In the text and Web channel it is mandatory, as this output will be used to display the navigation command so that the caller can activate it. In the voice and video channel, the available commands are typically explained to the caller within the welcome message, so the presentation outputs are not required.

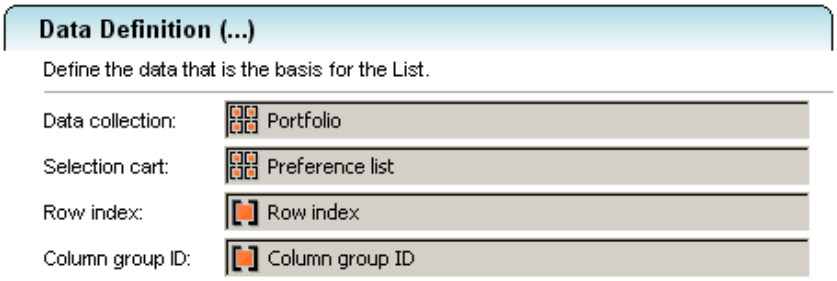


From the **Navigation direction** drop-down list at the top, select one of the available commands, and in the **Activation Output** or **Presentation Output** section define the desired output (see the Output object for more details). In addition, layers can be associated with individual navigation messages. This way, a navigation output can, for example, depend on the caller's current position in the list and be disabled if desired.

Copyright © 2001-2011 Voxeo Germany GmbH

Data Definition

The content of the List object is stored in a Collection object.



The **Data collection** field specifies the Collection object that holds the list data. The collection must consist of one or more rows, each holding one or more columns. The data columns can have arbitrary names. If direct access to rows is desired, add columns with the *name* attribute set to *grammarVoice* and/or *grammarDTMF* for each row. The content of these columns must be a comma-separated list of utterances that serve as the grammar to access this row. See below for an example.

Note: Direct row access is only supported in the voice and video channel.

The **Selection Cart** field allows you to specify a Variable or Collection object, which serves as a place in which to accumulate one or more selections of List items during the processing of the list. It can be used to implement a virtual “shopping cart” that is filled by the caller during the dialog. If a Variable object is specified, then a comma-separated list of *keys* (as given in column *name=”key”* of the data collection) is generated for all selected items. If a collection is specified, then the entire current row is copied from the data collection into the selection cart collection. This new collection can later serve as the data collection for another List object that e.g. presents all items from the shopping cart, to allow the caller to review the items and optionally select or delete some. If the selection cart collection is not empty when entering the list and does not have the same column layout as the data collection, then the server responds with an *Error - Internal* when trying to copy the row into the selection cart collection.
The selection cart is optional, but if defined, the data collection needs to contain a column named *key*.

For the stock portfolio example, the data collection might look as follows:

```
<root>
  <row>
       <col name=”key”>ibm</col>
       <col name=”grammarVoice”>ibm</col>
       <col name=”name”>IBM</col>
       <col name=”quote_today”>17.5</col>
       <col name=”quote_yesterday”>16.3</col>
       <col name=”news”>None.</col>
       <col name=”portfolio”>56</col>
  </row>
  <row>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
        <col name=”key”>apple </col>
        <col name=”grammarVoice”>apple, apple computers</col>
        <col name=”name”>Apple</col>
        <col name=”quote_today”>23.8</col>
        <col name=”quote_yesterday”>25.4</col>
        <col name=”news”>
               Apple will release their new operating system tomorrow.
        </col>
        <col name=”portfolio”>127</col>
    </row>
    <row>
        <col name=”key”>dell</col>
        <col name=”grammarVoice”>dell</col>
        <col name=”name”>Dell</col>
        <col name=”quote_today”>56.1</col>
        <col name=”quote_yesterday”>50.0</col>
        <col name=”news”>
               Dell’s newly introduced laptop line is flying off the
               shelves.
        </col>
        <col name=”portfolio”>83</col>
    </row>
</root>
```

In the **Row index** field, link a Variable, Script, Expression, or Layer object. The row index serves two purposes:

- When first entering the List object, or when re-entering it after a selection command has been processed, the index indicates the row in which to start the presentation by setting it to the numerical index of the row (starting with 1). If it is empty or has a value that is outside the range of rows within the data collection, the first row is presented.
- If a Variable object is used as row index, it holds the index of the row currently being presented, for referencing purposes in selection command processing or in column group or navigation message formatting. Note that this does not work for Script, Expression, or Layer objects.

This field is optional, and if not given, the presentation of the list starts with the first row.

The **Column group ID** behaves quite similarly:

- It allows to start presenting a row at a specific column group, e.g. when first entering a row or after a selection command has been processed. Column groups are identified by their ID specified in the **Content Formatting** section.
- If a Variable object is used, it contains the ID of the column group currently being presented. Again, this does not work for Script, Expression, or Layer objects.

Copyright © 2001-2011 Voxeo Germany GmbH

This field is also optional, and if not given, the presentation of each row starts with the first column group.

Content Formatting

The **Content Formatting** section defines any number of individual column groups, which form the logical entity that is presented to the caller and allows flexible formatting of the data to be read out. Note that *column group* is an abstract term; a column group does not necessarily need to be made up of actual columns from the data collection. It can consist of text (in all channels) or audio/video files (plus alternative text, in the voice and video channel only) defined in columns of the data collection, but also of existing objects that can be played back (such as Output, Audio, Video, Text, or Variable objects). When creating the List object, the designer needs to decide which columns from the data collection to present in what way. It is both possible to ignore certain columns of the collection, or put the content of several columns into one prompt, i.e. form one column group out of several columns. In case of a simple one-dimensional list, exactly one column group needs to be defined. In case of a two-dimensional table (N x M), the number of column groups corresponds to the horizontal dimension N of the resulting List object, whereas M is the number of rows of the data collection.

In the voice and video channel, it is possible for the caller to just listen to the list, without interacting with it. If that is the case and the check box **Enable auto-advance to the next column group** is selected (see Options), each column group is presented one after the other, in the order they are defined in the **Content Formatting** section. When having reached the end of a row, the first column group of the following row is presented next. This way, the list is traversed column group by column group, row by row. If a grammar is specified for a column group (see below) the caller can directly jump to this column group by uttering the corresponding command.

For the voice and video channel, barge-in for the outputs played within a column group can be controlled individually for each column group.



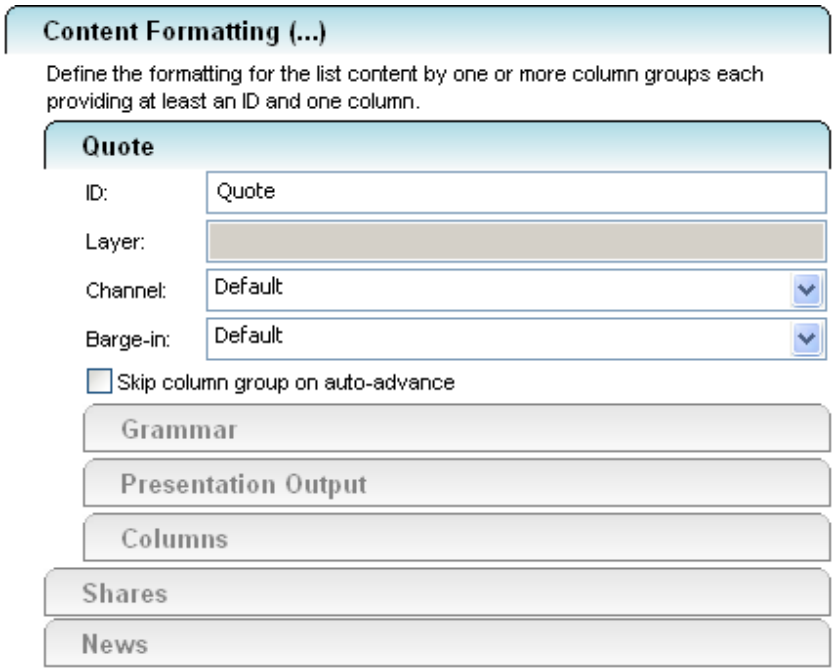
Note: When disabling barge-in on a column group, make sure the caller has sufficient time to make an input after the outputs for the column group have fully played, and before auto-advance to the next column group or row is activated.



Note: In the text and Web channel, neither auto-advancing nor directly jumping to a column group is supported.

The **Content Formatting** section can hold any number of column group items each defining a column group.

Copyright © 2001-2011 Voxeo Germany GmbH



In each column group item the following properties can be specified:

Property	Description
ID	Mandatory parameter to identify the column group. If a column group ID variable is defined in the **Data Definition** section, it is this ID that is stored in the variable to track the current horizontal position in the list during call time.
Layer	Layer definition for the column group. This layer is evaluated whenever the column group is entered, so that the presentation of the column group can be disabled or enabled dynamically during the session, e.g. after processing a selection command.
Channel	Channel setting for the column group. The column group is only presented when the channel is set to the currently active channel at call time, or if it is set to *Default*, in which case it is active in all channels.
Barge-in	Indicates whether barge-in is allowed within the column group or not. The default value *Default* specifies that the dialog context setting is applied. Barge-in only applies to the voice and video channel.
Skip column group on auto-advance	(This check box only applies to the voice and video channel. If it is selected, the column group is not presented when auto-advancing through the row. If a grammar is specified for the column group, the caller can still access it by uttering the corresponding command (see next item).

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Grammar	The grammar specifies the utterances the caller can make to jump to this column group. This grammar is optional. In the voice and video channel, if no grammar is defined, an automatic grammar is generated, based on the mandatory ID. The grammar does not need to fill any slots.
Presentation Output	An output to define how the column group is presented to the caller. This is primarily relevant in combination with the **Grammar**.
Columns	The list of columns forming the column group. Note that *column* is an abstract term and not necessarily associated with an actual column from the data collection. It is possible to combine content (text, audio, video) coming from the list with other data present in the dialog, like variable content or Output objects etc. See below for details on the **Columns** subsection.

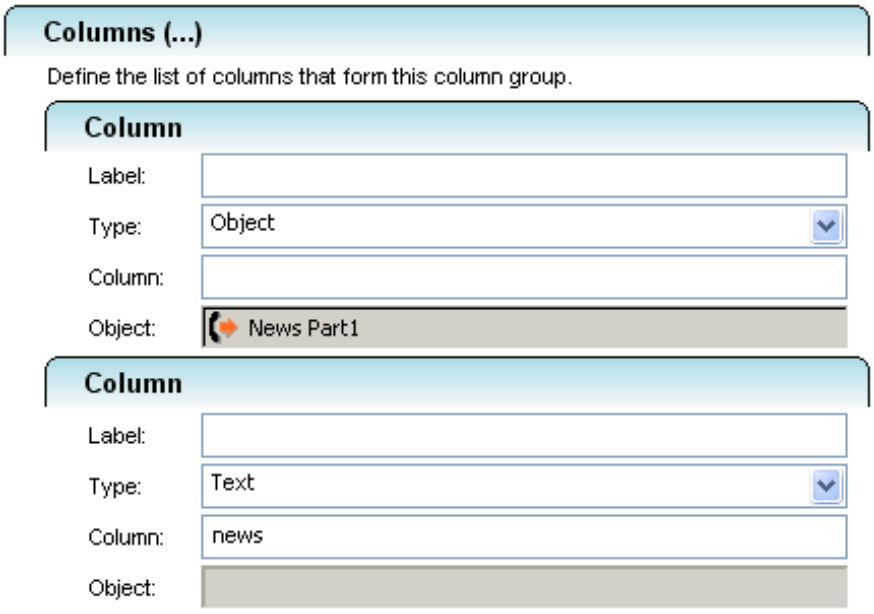
Columns

The **Columns** subsection can hold one or more items, which together form the prompt for this column group.

As an example, a column group can consist of an Output object (*Today's quote is…*), followed by content from the data collection (*56.1*). In this case, the **Columns** section would consist of two items.

The items of the **Columns** section are presented in the order they are defined.

As with other object items (e.g. Output items) you can also re-order whole **Column Groups** as well as items in the **Columns** section within the editor. Refer to Chapter 6 – *Object Editors* in the *Desktop for Eclipse Guide* or *Desktop for Web Guide* for information on how to re-order object items.



Copyright © 2001-2011 Voxeo Germany GmbH

Each item contains the following fields:

Label
An optional label for this item.

Type
A drop-down list specifying the type of content for this item. It can be one of *Audio, Video, Text*, *Object*, and *Object reference*.

- *Audio* indicates an audio filename coming from the data collection. This type is only supported in the voice and video channel. The **Column** field specifies the column the filename is retrieved from, and the **Object** field lets you optionally link a Resource Locator object that is used to locate the audio resource (in case the column from the data collection does not already contain a path for the file).
  The column can optionally hold alternative text that will be read out if the audio file cannot be played back. Alternative text is attached to the filename using the separator ";" (semi-colon). Any text coming after the first occurrence of ";" is taken as the alternative text (this includes other semi-colons), whereas any text coming before is taken as the filename. If the `<col>` starts with a semi-colon, i.e. no filename is given, the alternative text will be rendered into the VoiceXML document with no surrounding `<audio>` element. If no semi-colon is present, the content of the `<col>` element is taken as the filename.

  Example:
  ```
  <row>
    ...
    <col name="news">ibmnews.wav;There are currently no news
  on IBM</col>
    ...
  </row>
  ```

  Given that no Resource Locator object is linked in the **Object** field, this will be rendered as
  ```
  <audio src="ibmnews.wav">There are currently no news on
  IBM</audio>
  ```
- *Video* indicates a video filename coming from the data collection. This type is only supported in the video channel. The **Column** field specifies the column the filename is retrieved from, and the **Object** field lets you optionally link a Resource Locator object that is used to locate the video resource (in case the column from the data collection does not already contain a path for the file).
  Note that the server will always include the current video type setting coming from the dialog context in the generated VoiceXML code.
- *Text* indicates textual data coming from the data collection. The **Column** field specifies the column the data is retrieved from, and the **Object** field lets you optionally link a Format object to be applied to the data. This way, data from the collection can flexibly be read out via TTS or TTA (prerecorded audio files) in the voice and video channel, or presented in a formatted way in the text and Web channel.
- *Object* indicates an object within the project, which is specified in the **Object** field. Note that in this case the column definition does not refer to a value from the data collection. *Object* is meant to allow objects like Output as part of a column group. If the object referenced here has pre- or post-processing, this is ignored.
- *Object reference* indicates an object reference (via a reference ID) coming from the data collection. This allows you to process an object from the current project, driven by the data collection. The difference between *Object* and *Object reference* is that with *Object*, you link an existing object statically into the List object, whereas with *Object reference*, the object to be processed is dynamically defined

Copyright © 2001-2011 Voxeo Germany GmbH

based on the content of the data collection. The **Column** field specifies the column from which the object reference is retrieved, and the **Object** field lets you optionally specify a Format or Resource Locator object to be applied to the referenced object. If a Format object is used, it overrides any formatting that might be attached to the referenced object itself. Likewise, if a resource locator is given, it overrides any resource locator that might be attached to the object. Note that if no object with the given reference ID exists, the server responds with an *Error - Internal*. If an object with the given reference ID exists in your project but is not referenced anywhere in the dialog flow, add it to the Reference Library of the Module object to make sure it is loaded into the cache of the server on deployment.

Object
To link a Resource Locator object, in case the **Type** field is set to *Audio* or *Video*, and to link a Format object for *Text* or *Object reference*.
If **Type** is set to *Object*, link any object that can be played back (Output, Variable, Expression, Collection, Audio, Video, Text, Silence, and Script).

Column
A text field that contains the name of the data collection column providing the data for this element. If **Type** is set to *Object*, this field is ignored.

Selection Commands

The **Selection Commands** section defines any number of commands that can be performed on a List item. Examples are to select an entry from a list in a shopping application or to delete an e-mail from your inbox in an e-mail reader application.



In each **Selection Command** section the following properties can be specified:

Property	Description
Label	An optional label to identify the selection command.
Layer	A layer definition that is evaluated every time a new column group is entered during call time, so that selection commands can be enabled and disabled dynamically.

Copyright © 2001-2011 Voxeo Germany GmbH

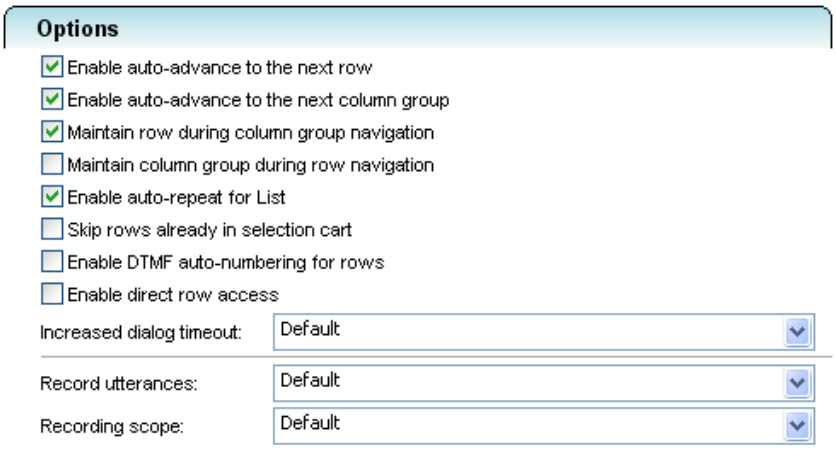
Property	Description
Action	This field is used to define special commands to add or remove items from the selection cart (if defined). If the selection command is not dealing with the selection cart, leave this setting at *None*. For a command that adds the current List item to the selection cart (if defined), set this field to *Add item to selection cart*. When the caller activates this command, the server adds the current item to the cart. You can optionally link any object in the **Object** field, e.g. an Output object presenting a message to confirm the selection. For a selection command that removes the current item from the selection cart (given it is already part of the cart), set this field to *Remove item from selection cart*. When the caller activates this command, the server removes the item that the caller is currently on from the selection cart. If the current item was not yet part of the selection cart, no action is performed and the current item is played again.
Exit list	If set to *True*, the list processing ends after the selection command has been processed and the dialog proceeds with the next object in the dialog flow. If set to *False* (the default), the List object proceeds with the next row.
Object	The object to be processed upon activation of the selection command. If the **Action** field (see above) is set to *Add item to selection cart* or *Remove item from selection cart*, this field can optionally be left empty.
Grammar	A grammar to define the input that activates this selection command. If no grammar is given, an automatic grammar is generated based on the label, or, if not given, the name of the object specified in the **Object** field (automatic grammars are only generated in the voice and video channel). If the latter is not defined, the server responds with an *Error – Internal*. No slots need to be filled by this grammar. For more information on how to define a grammar, refer to the Grammar object in this *Object Reference*.
Presentation Output	An output to define how the selection command is presented to the caller. In the text and Web channel it is mandatory, as this output will be used to display the selection command so that the caller can activate it. In the voice and video channel, the available commands are typically explained to the caller within the welcome message, so the presentation outputs are not required.

Tip: It is possible to modify the row index and column group ID (see Data Definition) while processing a selection command (e.g. by using Expression objects), so that flexible navigation commands can be implemented.

Copyright © 2001-2011 Voxeo Germany GmbH

Options

In the **Options** section, the editor offers several check boxes that mainly influence the list navigation behavior.



Enable auto-advance to the next row (selected by default)
This option only applies to the voice and video channel.
If selected, the first column group in the next row of the data collection is presented in the absence of any caller input at the end of the current row. When in the last row, the **End of List** message is played. The time to wait until the list advances to the next row can be specified by setting the *Input - Speech timeout* parameter in the Tuning definition of the List object.

Enable auto-advance to the next column group (selected by default)
This option only applies to the voice and video channel.
If selected, the list automatically advances to the next column group in the absence of any caller input while or after playing the current column group. The time to wait until the List object automatically advances to the next column group can be influenced by adding Silence objects to the output definition of the column group (see Content Formatting). When the check box **Enable auto-advance to the next row** is clear and the last column group of the current row is played, the presentation starts again from the first column group.

Maintain row during column group navigation (selected by default)
If selected and the caller activates a horizontal navigation command to switch column groups (*nextCG* or *previousCG*, when at least two column groups exist in the list definition) and passes a row border (i.e. first or last column group of a row), the presentation will stay in the current row. Example: If the caller activates *nextCG* while being in the last column group of a row, the first column group of the same row is being played next. Vice versa, activating *previousCG* on the first column group will jump to the last column group of the same row.
If this check box is clear, the List object always jumps to the next or previous row when passing row borders. Example: If the caller activates *nextCG* while being in the last column group of a row, the first column group of the next row is being played next. Vice versa, activating *previousCG* on the first column group will jump to the last column group of the previous row. This approach allows to offer the caller a navigation command that walks through a two-dimensional list from the "top left" (1, 1) to the "bottom right corner" (n, n).

Maintain column group during row navigation (clear by default)
If selected and the caller activates a vertical navigation command to switch rows (*nextRow*, *previousRow*, *start* etc.), the presentation of the target row starts at the current column group (the one the caller currently hears) instead of the first one. This

Copyright © 2001-2011 Voxeo Germany GmbH

allows selecting one particular item of interest for all rows, e.g. *News* in the stock portfolio example (see Dialog Flow Scenario), and navigating between these (staying in one column group of the List object). If this check box is clear, the List object always jumps back to the first column group when switching rows.
If there is only one column group defined in the List object, this option has no effect.

Enable auto-repeat for List (selected by default)
If selected, the List object automatically returns to the first List item when the last item has been presented and no caller input has occurred. Before doing so, the **End of List** output is presented (if defined).

Skip rows already in selection cart (clear by default)
This option allows the use of the selection cart as a negative list for the data collection. If selected and a selection cart is defined, rows that are already in the cart are ignored, i.e. not presented anymore. If all available rows in the data collection are also in the cart (making the two collections identical) the List object is left and the dialog flow proceeds with the next object.

Enable DTMF auto-numbering for rows (clear by default)
This option only applies to the voice and video channel.
If selected, rows can be accessed directly by entering the row index via DTMF (e.g. typing "*3* " to jump to the third row, or "*1 4*" to jump to the fourteenth row). This is only available when the option **Enable direct row access** is selected.

Enable direct row access (clear by default)
This option only applies to the voice and video channel.
It allows the caller to jump to a particular row directly by saying what is defined in the *grammarVoice* or *grammarDTMF* columns of the data collection (see Data Definition for an example of *grammarVoice*). If these grammars are not available, a DTMF grammar is automatically created based on the numerical row index, in case the check box **Enable DTMF auto-numbering for rows** is selected.

Increased Dialog Timeout specifies a timeout value that is applied for this object in case it is higher than the current dialog timeout in a session. If it is lower, the setting is ignored. It is mainly applicable in the text and Web channel, where the dialog timeout of a service should be set to a relatively low value, e.g. 40 seconds, to keep the number of concurrent sessions low, as they usually run into the server timeout due to technical reasons. Use this setting to increase the timeout temporarily for objects that are likely to require more time to respond to, e.g. when the caller has to type in an address.

Record utterances specifies whether utterances made in this object are to be recorded (*True*) or not (*False*), given that utterance recording is switched on for this session. *Always* indicates that utterances in this object are always recorded when utterance recording is enabled on the service, regardless of statistical filtering. By default the value is *Default*, indicating that the current value of the corresponding dialog context setting RECORDUTTERANCES is used.

Recording scope defines the scope in which recordings will be made. If set to *Recognition*, only those utterances will be recorded that were successfully recognized (including the activation of hyperlinks). If set to *No Match*, only those utterances will be recorded that triggered a No Match event. If set to *All*, any recording will be stored. If set to *Default*, the current value of the corresponding dialog context setting RECORDINGSCOPE is used. By default the value is *Default*.

Due to the nature of recordings, the feature of recording utterances is only available in the voice and video channel. For more information on this refer to Chapter 6 – *Recording of Utterances* in the *Deployment Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

VoiceObjectsXML Definition

The List object is represented by the VoiceObjectsXML element `<list>`. It has fourteen attributes and fourteen groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<list>` element uses the embedded `<contentFormatting>`, `<navigationMessages>`, and `<selectionCommands>` elements.

List

Attributes

- **`autoAdvanceColumnGroup`**
  Defines whether the list should automatically advance to the next column group in the current row if no caller input occurs. Either *true* or *false*. If not specified, defaults to *true*.
- **`autoAdvanceRow`**
  Defines whether the list should automatically advance to the next row if no caller input occurs. Either *true* or *false*. If not specified, defaults to *true*.
- **`maintainColumnGroup`**
  Defines whether to maintain the current column group when navigating between rows by explicit caller input. Either *true* or *false*. If not specified, defaults to *false*.
- **`maintainRow`**
  Defines whether to maintain the current row when navigating between column groups by explicit caller input. Either *true* or *false*. If not specified, defaults to *true*.
- **`autoRepeat`**
  Defines whether the list loops around from the last column group of the last row to the first column group of the first row. Either *true* or *false*. If not specified, defaults to *true*.
- **`excludeCart`**
  Defines whether rows already in the selection cart should be excluded from presentation during auto-advance. Either *true* or *false*. If not specified, defaults to *false*.
- **`enableDirectRowAccess`**
  Defines whether direct access to individual rows is possible. Either *true* or *false*. If not specified, defaults to *false*. If it is set to true, note that the data collection needs to contain a column *grammarVoice* or a column *grammarDTMF* (or both).
- **`dataCollection`** [required]
  Reference to a Collection object containing the list data.
- **`rowIndex`**
  Reference to a Variable object that is used to maintain the index for the currently active row within the List object. May also be provided via a reference to an Expression or Script object, in which case the current position is not written back.
- **`columnGroupID`**
  Reference to a Variable object that is used to maintain the ID of the currently active column group within the current row. May also be provided via a reference to an Expression or Script object, in which case the current position is not written back.
- **`selectionCart`**
  Reference to a Variable or Collection object used to maintain the cart.

Copyright © 2001-2011 Voxeo Germany GmbH

- **recordUtterances**
  Defines whether to use utterance recording. One of *true*, *false*, *always*, *default*. If not specified, defaults to *default*.
- **recordingScope**
  Defines the scope for utterance recording. One of *all*, *noMatch*, *recognition*, *default*. If not specified, defaults to *default*.
- **increasedDialogTimeout**
  Defines the increased dialog timeout in seconds. If not specified, the setting from the dialog context is applied at call time.

Children

- **<expression usage="precondition">** or
  <variable usage="precondition"> or
  <collection usage="precondition"> or
  <script usage="precondition">
  Defines the precondition for the List object.
- **<sequence usage="preprocessing">**
  Defines the preprocessing sequence for the List object.
- **<output usage="listWelcome">**
  Defines the "list welcome" message.
- **<output usage="listBegin">**
  Defines the "begin of list" message.
- **<output usage="listEnd">**
  Defines the "end of list" message.
- **<output usage="listEmpty">**
  Defines the "no data" message.
- **<grammar usage="navigation">** [required]
  Defines the navigation grammar for the List object. Needs to fill a slot *sltNavigation* with the tokens *start*, *firstCG*, *firstRow*, *end*, *lastCG*, *lastRow*, *previous*, *previousCG*, *previousRow*, *next*, *nextCG*, *nextRow*, *repeat*, *stop*. (Note that it is possible to define a different slot name through a media platform driver if required).
- <**contentFormatting**> [required]
  Defines how to present the list content via column groups.
- <**navigationMessages**>
  Defines the set of navigation messages.
- <**selectionCommands**>
  Defines the set of selection commands.
- **<eventHandling>**
  Defines the event handling for the List object.
- **<customNavigation>**
  Defines the custom navigation settings for the List object.
- **<tuning>**
  Defines the tuning settings for the List object.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the List object.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<list name="Present health care plans" dataCollection="#Health care
plans" selectionCart="#Selected plan">


  <!-- Outputs -->
  <output usage="listWelcome">
    <outputItem occurrence="1">
      <text>Here is the list of our health care plans. You can
       browse through it by saying forward or backward. When you
       hear the plan you are interested in, say select. To return to
       the main menu, just say main menu.</text>
    </outputItem>
    <outputItem occurrence="2">
      <text>Here is a list of our health care plans.</text>
    </outputItem>
  </output>
  <output usage="listBegin">
    <outputItem>
      <text>You are at the beginning of the list. You cannot go back
       any farther.</text>
    </outputItem>
  </output>
  <output usage="listEnd">
    <outputItem>
      <text>This is the end of the list.</text>
    </outputItem>
  </output>
  <output usage="listEmpty">
    <outputItem>
      <text>We are not currently offering any health insurance
       plans. </text>
    </outputItem>
  </output>


  <!-- Navigation grammar -->
  <grammar usage="navigation">
    <grammarItem>
      <grammarDefinition mode="voice" location="#Grammar locator"
       file="navigation"/>
    </grammarItem>
  </grammar>


  <!-- Content Formatting -->
  <contentFormatting>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
  <columnGroup ID="Plan">
    <column type="text" columnName="plan"/>
  </columnGroup>
</contentFormatting>

<!-- Navigation Messages -->
<navigationMessages>
  <navigationMessage direction="next">
    <output usage=”activation”>
      <outputItem language="en-US">
        <text>Moving on to the next plan.</text>
      </outputItem>
    </output>
  </navigationMessage>
  <navigationMessage direction="previous">
    <output usage=”activation”>
      <outputItem language="en-US">
        <text>Backing up to the previous plan.</text>
      </outputItem>
    </output>
  </navigationMessage>
</navigationMessages>

<!-- Selection Commands -->
<selectionCommands>
  <selectionCommand action="add" exit="true">
    <grammar>
      <grammarItem>
        <grammarDefinition mode="voice" location="#Grammar
         locator" file="selection"/>
      </grammarItem>
    </grammar>
    <output name="Play health insurance fee">
      <outputItem>
        <text>The monthly fee for this plan is only </text>
        <expression function="index">
          <argument object="#Health care plans"/>
          <argument object="#Selected plan"/>
          <argument><value>PlanPrice</value></argument>
        </expression>
        <text> dollars.</text>
      </outputItem>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
      </output>
    </selectionCommand>
  </selectionCommands>
</list>
```

ContentFormatting

Children

- +<**columnGroup**>
  Defines the list of individual column groups.

Example

```
<contentFormatting>
  <columnGroup ID=”SingleTrack”>
    <column type=”object” property=”#Track intro 1”/>
    <column type=”audio” columnName=”artist”/>
    <column type=”object” property=”#Track intro 2”/>
    <column type=”audio” columnName=”track”/>
  </columnGroup>
</contentFormatting>
```

ColumnGroup

Attributes

- **ID** [required]
  A text string providing an ID for the column group.
- **layer**
  Defines the layer for the column group. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **channel**
  Defines the channel(s) for which this column group is valid. Can be *default*, *voice*, *video*, *text*, *web*, *voiceVideo*, or *textWeb*. If not specified, defaults to *default*.
- **bargein**
  Defines the barge-in behavior of the column group. Can be *true*, *false*, or *default*. If not specified, defaults to *default*.
- **skip**
  Defines whether this column group should be skipped in auto-advance. Either *true* or *false*. If not specified, defaults to *false*.

Children

- **<grammar>**
  Defines the grammar for direct access to this column group.
- **<output>**
  Defines the presentation output for this column group.

Copyright © 2001-2011 Voxeo Germany GmbH

- +<column>
  Defines the list of columns in this column group.

Example

```
<columnGroup ID=”SingleTrack”>
  <column type=”object” property=”#Track intro 1”/>
  <column type=”audio” columnName=”artist”/>
  <column type=”object” property=”#Track intro 2”/>
  <column type=”audio” columnName=”track”/>
</columnGroup>
```

Column

Attributes

- **label**
  A text string providing a name for the column.
- **type** [required]
  Defines the type of this column. One of *object*, *text*, *audio*, *video*, *reference* (see type description above).
- **property**
  Defines a reference to an object. The legal object types vary depending on the selected type.
  If attribute `type` has the value *object*, the object itself can also be defined as a child.
- **columnName**
  Specifies the column name in the data collection from which to retrieve data. The interpretation of this data depends on the selected type.

Children

- **<expression>** or
  <script> or
  <variable> or
  <collection> or
  <output> or
  <layer> or
  <silence> or
  <audio> or
  <text> or
  <video>
  Defines an object to be played. Only valid if attribute `type` has the value *object*.

Example

```
<column type=”text” columnName=”author”/>

<column type=”audio” property=”#Music locator” columnName=”track”/>

<column type=”object”>
  <output>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
      <outputItem>
        <text>Now moving on to the next entry.</text>
      </outputItem>
  </output>
</column>
```

NavigationMessages

Children

- *<**navigationMessage**>
  Defines the list of navigation messages; at most one for each navigation direction.

Example

```
<navigationMessages>
  <navigationMessage direction=”next”>
    <output usage=”activation”>
      <outputItem>
        <text>Now moving on to the next entry.</text>
      </outputItem>
    </output>
  </navigationMessage>
</navigationMessages>
```

NavigationMessage

Attributes

- **direction** [required]
  Specifies the navigation direction to which this navigation message applies. Possible values are *previous*, *previousRow*, *previousCG*, *next*, *nextRow*, *nextCG*, *repeat*, *start*, *firstRow*, *firstCG*, *end*, *lastRow*, *lastCG*, *stop*, *autoAdvance*.
- **layer**
  Defines the layer for the selection command. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.

Children

- **<output usage=”activation”>**
  Defines the output that is played when the navigation direction is activated.
- **<output usage=”presentation”>**
  Defines the presentation for this navigation direction.

Example

```
<navigationMessage direction=”next”>
  <output usage=”activation”>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
    <outputItem>
      <text>Now moving on to the next entry.</text>
    </outputItem>
  </output>
</navigationMessage>
```

SelectionCommands

Children

- *<**selectionCommand**>
  Defines the list of selection commands corresponding to the commands that can be performed on list cells.

Example

```
<selectionCommands>
  <selectionCommand action=″add″>
    <output bargein=″false″>
      <outputItem language=″en-US″>
        <text>The item has been added to your shopping cart.<text>
      </outputItem>
    </output>
  </selectionCommand>
</selectionCommands>
```

SelectionCommand

Attributes

- **label**
  A text string providing a name for the selection command.
- **layer**
  Defines the layer for the selection command. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **exit**
  Either *true* or *false*. Indicates whether the processing of the List object is terminated after processing the selection command. If not specified, defaults to *false*.
- **action**
  Indicates whether the selection cart should be modified. Legal values are *none*, *add*, *remove*. For *add*, the key for the current list row is added to the cart variable or collection before processing the selection command. For *remove*, the key for the current list row is removed from the cart variable or collection before processing the selection command. For *none*, nothing is done with the selection cart.
  If not specified, defaults to *none*.

Copyright © 2001-2011 Voxeo Germany GmbH

Children

- <**grammar**> [required]
  Defines the activation grammar for the selection command.
- **<output usage="presentation">**
  Defines the presentation for the selection command.
- One of the following:
  <module>, **<input>**, **<output>**, **<sequence>**, **<menu>**, **<confirmation>**, **<list>**, **<osdm>**, **<plugin>**, **<connector>**, **<script>**, **<log>**, **<if>**, **<case>**, **<goto>**, **<loop>**, **<expression>**, **<recording>**, **<transfer>**, **<pause>**, **<exit>**
  Defines the object to be processed when the selection command is activated.

Example

```
<selectionCommand action="add">
  <output bargein="false">
    <outputItem language="en-US">
      <text>The item has been added to your shopping cart.<text>
    </outputItem>
  </output>
</selectionCommand>
```

Usage Examples

Usage examples for the List object are:

- Presenting a list of items fitting a certain description (e.g. top ten tracks, books by a certain author, etc.) with the option to place one or more items in a shopping cart.
- Presenting a stock portfolio with quotes, news, etc. with the option to perform actions (buy, sell, etc.) on the stock.
- Presenting the weather forecast for a city for the next several days, browsable by days.
- Presenting traffic information for a list of major highways.
- Presenting sports score tables.
- Implementing an e-mail reader that allows reading an e-mail, forward it, reply to it, delete it, etc.

Object Interoperability

The following table contains all object types that can reference a List object:

Icon	Object Name	Use Case Example
	Module	A List object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, a List object is used within a Sequence object in a dialog flow.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Object Name	Use Case Example
	Menu	A List object can be used as the destination object within a Menu item.
	Confirmation	A List object can be used as the destination object within a Correction item inside a Confirmation object.
	If	A List object can be within either the THEN item or the ELSE item of an If object.
	Case	A List object can be used in any WHEN item of a Case object.
	Loop	A List object can be used within the embedded sequence of a Loop object.
	Goto	A List object can be referenced via a Goto object.
	Hyperlink	A List object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a List object should describe which list of data is presented. The short description should contain a brief explanation of the caller's options. The table below lists three examples:

Name	Description
 Top Ten Tracks	Plays the current list of top ten tracks. Caller can add tracks to a shopping cart.
 Stock Portfolio	Presents the caller's stock portfolio with today's and yesterday's quotes, the number of shares owned, and news about the company.
 Weather Forecast	Presents the weather forecast for the next three days, browsable by day.

Copyright © 2001-2011 Voxeo Germany GmbH

Log

Overview

The Log object is a configurable dialog component that writes application-level logging information. This information may be written to a log file, to the media platform (typically some call detail record file), to a custom database, to the system database, or to the OSDM log, in case OSDM objects are used in the application. Logging information may contain arbitrary dynamic content such as caller input, data retrieved from back-ends, etc. It may be written out at any point during a session, or after the session has ended.

The Object Definition below covers the configuration of the Log object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Log object belongs to the object category Resources.

Dialog Flow Scenario

The following dialog flow example describes a simple Log object from the Prime Insurance Portal:

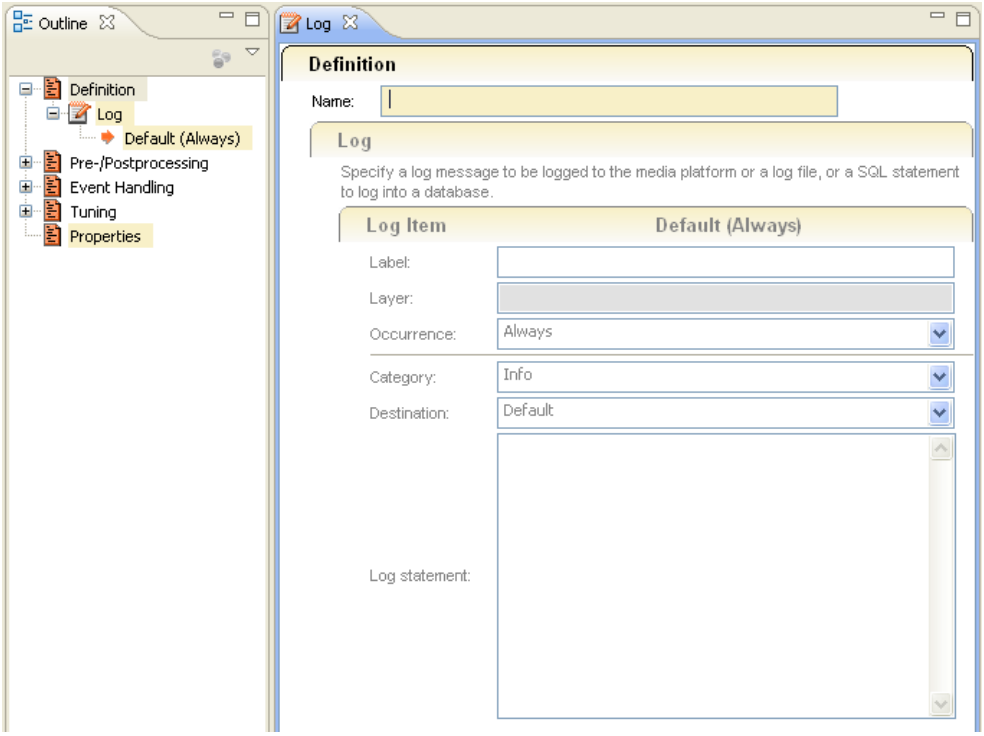
Object – Caller	Dialog Flow
	*Please say the make and model of your car*
 Caller	*Ford Thunderbird*
	Logs “Make = Ford / Model = Thunderbird” to a log file

Object Definition

The **Definition** of the Log object provides one section:

- Log
  To define the set of Log items.

Copyright © 2001-2011 Voxeo Germany GmbH



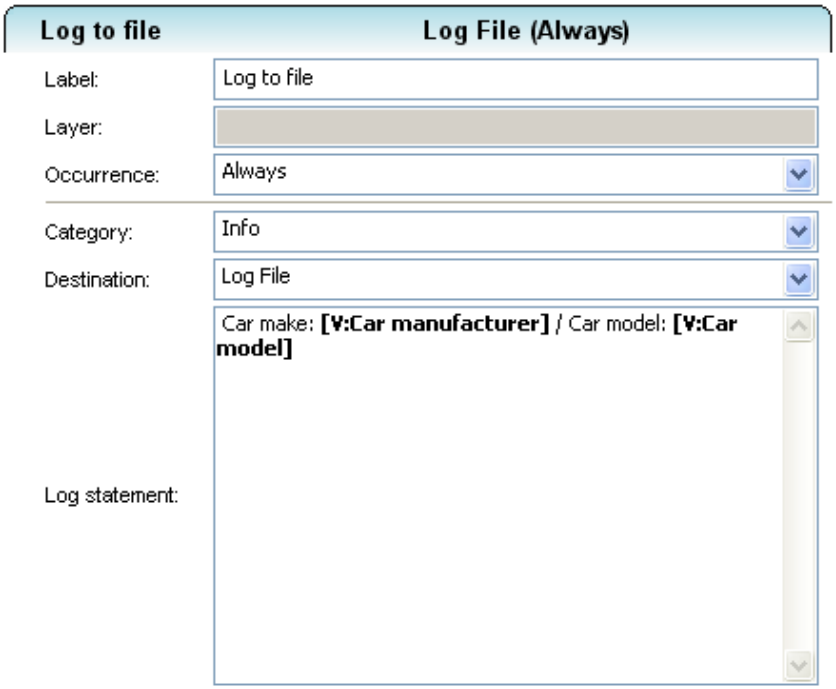
For further details regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Tuning*, and *Properties* in this *Object Reference*.

Log

The **Log** section contains at least one **Log Item** that writes log information to one of various configurable destinations. Different log items within the same **Log** object may log to different destinations. The log information can be a simple text or a combination of text and/or Variable, Collection, Expression, Script, and Layer objects.

A log definition may contain any number of individual Log items. If it has multiple Log items, they are processed in the order in which they are specified.

Copyright © 2001-2011 Voxeo Germany GmbH



Except of the actual log statement described below, for each Log item the following properties can be specified:

Property	Description
Label	Optional parameter to identify the Log item in a list.
Layer	Optional parameter to indicate the layer that the Log item belongs to. For more information on layers, see the Layer object in this *Object Reference* or Chapter 7 – *How to Use Layers* in the *Design Guide*.
Occurrence	Enables you to log different information depending on how often the Log object has already been visited. *Always* means that regardless of the number of previous visits, the respective Log item is always processed. *Only once* means that the Log item is only processed the first time the Log object is visited. And *If >= N* means that the Log item is processed only if this is at least the Nth time that the Log object has been visited. Note that even if no Log item remains, the Log object's postprocessing (if defined) will still be processed.
Category	Identifies the message category that the respective Log item belongs to. Categories are intended to provide a filtering mechanism by which designers can classify individual log messages. Some messages may only be relevant during development, while others are important in a deployed service for statistics or billing purposes. Only Log items that belong to categories activated in the corresponding service are processed at call time. This makes it very easy to enable or disable groups of messages without actually having to modify the application itself. For more details, see Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Destination	Indicates where the log information is written to. The following destinations are available (see also Usage Examples): **Default** Indicates that the default destination defined in the corresponding service is used. For more details, refer to Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*. **Custom DB** Indicates that the Log item writes to a custom table in a database. When using this destination, the Log item needs to contain SQL code appropriate for the database system used. The SQL is passed on to the database after all object references within it have been evaluated. Note that even though no explicit restriction is placed on the SQL code, only INSERT or UPDATE statements are recommended as there is no mechanism to capture return data provided by SELECT statements. Only a single SQL statement may be provided per Log item. If multiple statements are required, these need to be split over multiple Log items that are processed in sequence. The SQL statement must **not** be terminated by a semicolon. For additional information on how to configure the Custom DB logging database connection, refer to Chapter 1 – *Advanced Configuration of VoiceObjects* in the *Administration Guide*. **Log File** Indicates that the Log item writes to a log file. A separate log file is created for each service. These files can be accessed via the **Control Center** as described in Chapter 3 – *Managing Servers and Services* in the *Deployment Guide*. **Log OSDM** Indicates that the Log item writes to the OSDM log, in case the application makes use of OSDM objects and OSDM logging is activated and defined properly in the corresponding Service object. For more information on this, refer to Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*. **Media Platform** Indicates that the Log item writes to the media platform log by using the VoiceXML `<log>` tag. Any information sent to this destination is included in the VoiceXML page rendered by the server. It can therefore be viewed in the Debug Viewer and in session tracing files, making this a useful destination for debugging with or without a media platform. **System DB** Indicates that the Log item writes to the system database; the information is stored at the end of the call. There is a buffer of at most 4000 characters per dialog to which new Log items are appended. For additional information on how to configure the System DB logging database connection, refer to Chapter 1 – *Advanced Configuration of VoiceObjects* in the *Administration Guide*. For examples of how to use the different logging destinations, see the Usage Examples paragraph below.

Copyright © 2001-2011 Voxeo Germany GmbH

Log items are processed in the order in which they are defined in the Log object. If one of them encounters problems, the others are still processed. Problems may occur e.g. if a database cannot be reached, the SQL code has an error, or a file cannot be written to. The event *Error Internal* is generated, so the problem can be handled as is appropriate for the application. For more details, refer to *Event Handling* in this *Object Reference*.

Note: If a problem occurs within any of the Log items in a Log object, then no VoiceXML code can be rendered by the Log object. This implies that none of the Log items with destination *Media Platform* will be able to write out their messages to the media platform. It is therefore recommended to keep Log items with destination *Media Platform* in a separate Log object.

Log statement

A Log item can be defined by entering plain text in the text area or by inserting new or existing Variable, Collection, Expression, Script, and Layer objects. For more information on how to do this, refer to Chapter 6 – *Object Editors* in the *Desktop for Eclipse Guide* and *Desktop for Web Guide*, respectively.

To insert a new object, drag the appropriate object type from the **Object Palette**. In Desktop for Web this is done via the context menu.

Tip: To simplify the mapping to the tables of the Infostore Repository you can use wildcards inside a log statement that represent key values from tables, like dialog ID, CRM ID, the ID of the active business task, etc. For a list of these wildcards, refer to Chapter 1 – *Welcome to Infostore* in the *Infostore Guide*.

Usage Examples

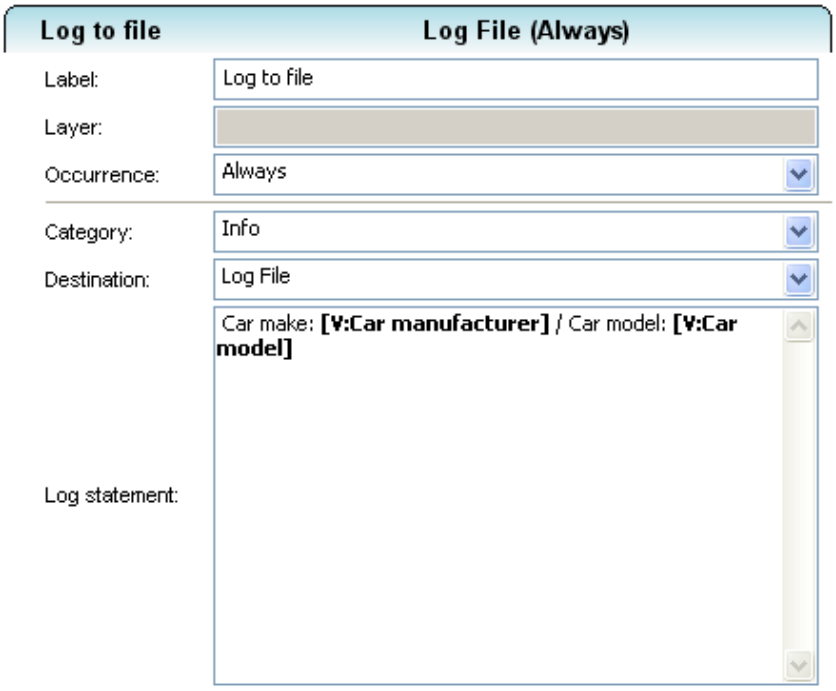
The following examples show how information entered by the caller, in this case a car make and model, can be logged to the various available destinations.

Log to a file

The default channel for logging is *Log File*, which writes to a service-specific file that can be accessed via the **Control Center** described in Chapter 3 – *Managing Servers and Services* in the *Deployment Guide*.

The Log item would be defined as follows:

Copyright © 2001-2011 Voxeo Germany GmbH



At call time, this creates an entry in the log file <servicename>_Service.log as follows:

```
[Category: INFO]
[Log Name: Log car data for dialog analysis]
[Label: Log to file]
[Ref ID: 0.129.129:OVAPac16172d00000000f9fb1f02ee_BVO_Log]
[VSN: prime_insurance]
[Dialog ID: OVAPac16172d0000000000000076000000fa08fc0de6]
[TS: 2004-01-12 08:45:46,736]
[DNIS: 022113542765]
[ANI: 0173764658477]
Car make: Chevrolet / Car model: Impala
```

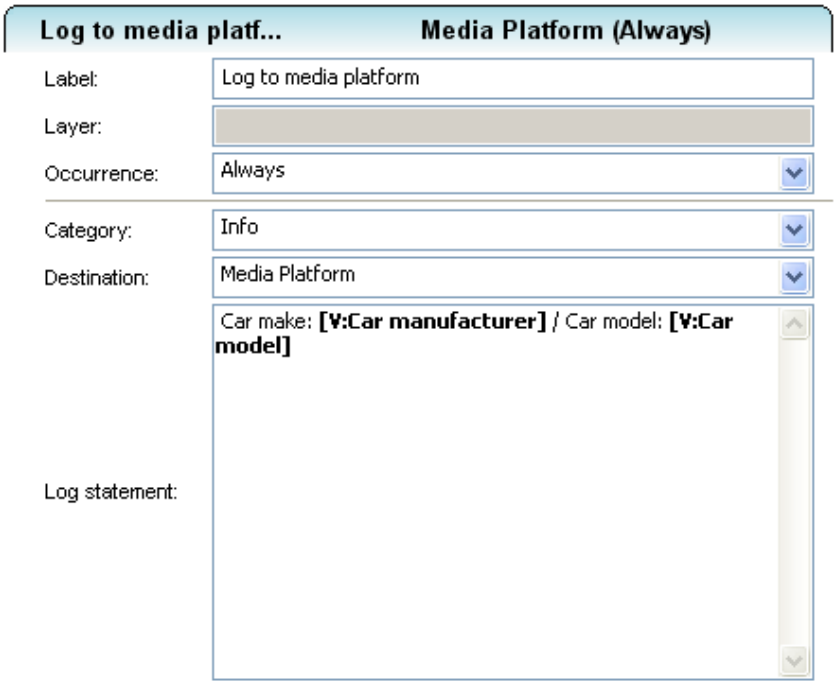
(Additional line breaks have been added for readability.)

Log to the media platform

Sometimes it is desirable to store logging information together with other information logged by the media platform itself. This can be accomplished using the *Media Platform* destination, which makes use of the VoiceXML element `<log>`.

The Log item must be defined as follows:

Copyright © 2001-2011 Voxeo Germany GmbH



The VoiceXML code generated at call time looks as follows:

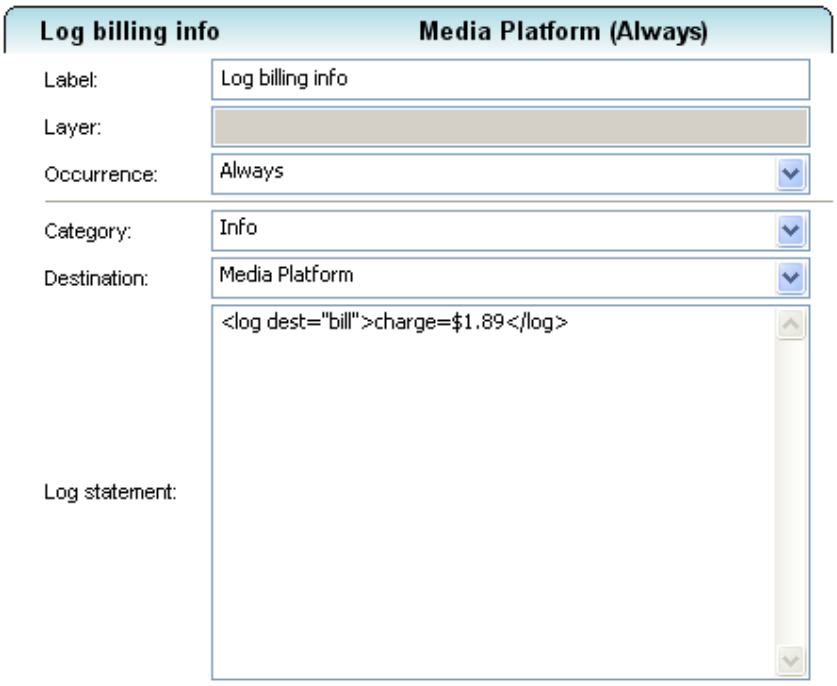
```
<log>
        <![CDATA[
                [Category: INFO]
                [Log Name: Log car data for dialog analysis]
                [Label: Log to file]
                [Ref ID:
                0.129.129:OVAPac16172d00000000f9fb1f02ee_BVO_Log]
                [VSN: insurance]
                [Dialog ID: OVAPac16172d00000000000026300000000fb15806e33]
                [TS: 2004-03-04 12:29:23,363]
                [DNIS: 022113542765]
                [ANI: 0173764658477]
                Car make: Ford / Car model: Mustang
        ]]>
</log>
```

(Additional line breaks have been added for readability.)

Information written to the *Media Platform* destination can also be viewed in the Debug Viewer and in session tracing files, making this a useful destination for debugging with or without a media platform.

When using the *Media Platform* destination, it is also possible to use a custom-coded inline `<log>` tag that is simply copied into the VoiceXML code. This allows designers to make use of custom attributes to the `<log>` tag, which some media platforms provide.

Copyright © 2001-2011 Voxeo Germany GmbH



The example shown here makes use of the custom attribute *dest,* which defines a destination, in this case *bill* for billing.

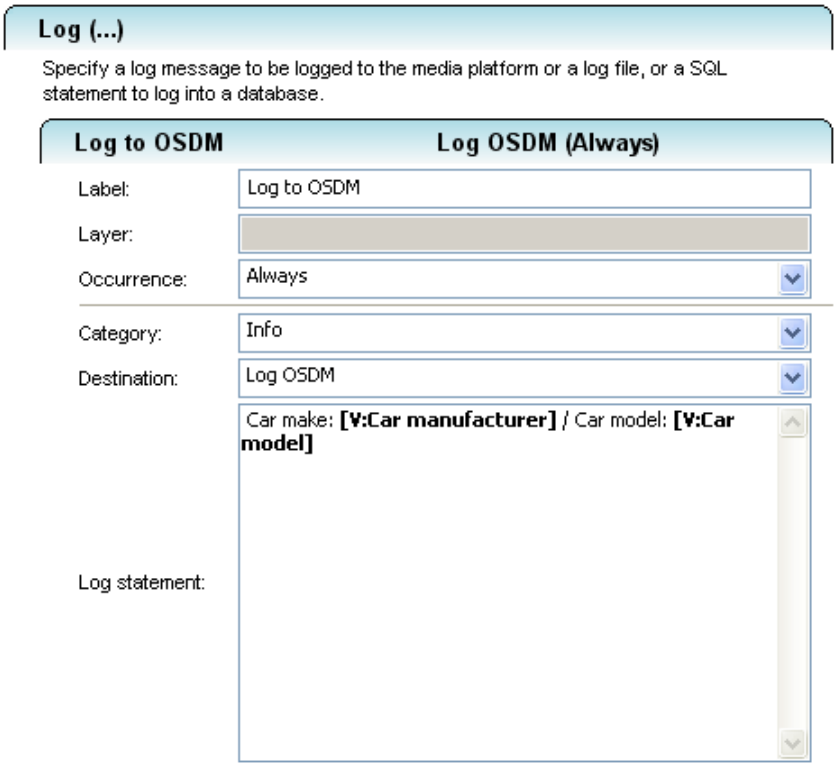
For more information on where the data provided within the `<log>` element is stored and how to access it, refer to the documentation of the respective media platform.

Log to the OSDM log

When your application makes use of OSDM objects, you can use the Log object to log data to dedicated log files that the OSDM server maintains. Make sure to activate OSDM logging and to set the correct logging destination in the corresponding Service object.

The Log item must be defined as follows:

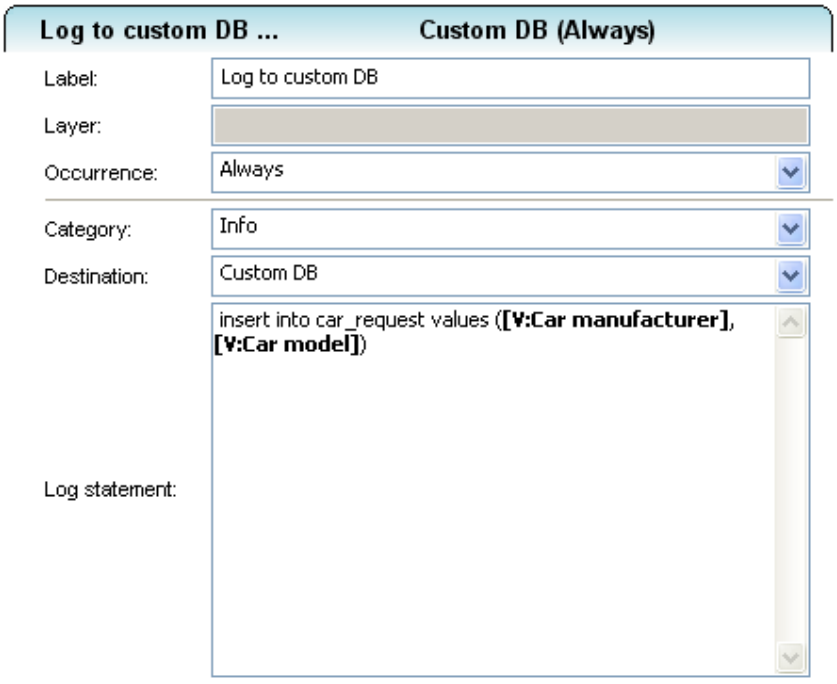
Copyright © 2001-2011 Voxeo Germany GmbH



Log to a custom database

Subsequent reporting is often easiest if the log data is already available in a database. One way of doing this is the *Custom DB* destination, which writes to custom-defined database tables.

The Log item must be defined as follows:



Copyright © 2001-2011 Voxeo Germany GmbH

Note that when using the *Custom DB* destination, the respective tables need to be created beforehand, and appropriate SQL code needs to be provided within the Log object. Only a single SQL statement may be provided per Log item, and there must **not** be a semicolon at the end of the statement.

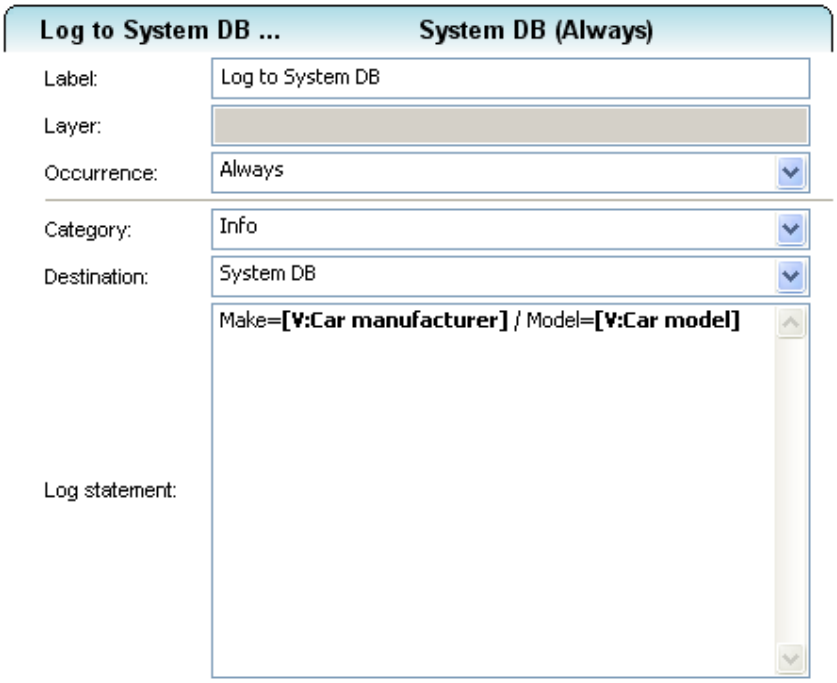
In order for Custom DB logging information to be written out, the **Enable Custom DB logging** check box in the respective service must be selected. In addition, the server hosting this service must have its *Custom DB logging* option enabled. Refer to **Control Center** in Chapter 3 – *Managing Servers and Services* in the *Deployment Guide* for a detailed description of how to enable and adjust the Custom DB logging status.

Note: Infostore is required for Custom DB logging. For further information, see the *Infostore Guide*.

Log to the system database

The other way of logging to a database is the use of the *System DB* destination. It writes out the log data as part of the System DB logging, which Infostore writes for each call (if so configured).

The Log item must be defined as follows:



The system database provides a buffer for these Log items of at most 4,000 characters per call. At the end of the call, this buffer is stored in the column *DLG_LOG* within the Infostore Repository. If the limit of 4,000 characters is exceeded during call time, any additional attempts to write to the system database are ignored. Note that the individual Log items are simply appended to each other. Use custom separators to distinguish between individual Log items.

In order for System DB logging information to be written out, the **Enable System DB logging** check box in the respective service must be selected. In addition, the server hosting this service must have its *System DB logging* option enabled. Refer to **Control**

Copyright © 2001-2011 Voxeo Germany GmbH

Center in Chapter 3 – *Managing Servers and Services* in the *Deployment Guide* for a detailed description of how to enable and adjust the System DB logging status.

Note: Infostore is required for System DB logging. For further information, see the *Infostore Guide*.

VoiceObjectsXML Definition

The Log object is represented by the VoiceObjectsXML element `<log>`. It has six groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<log>` element uses the embedded `<logItem>` element.

Log

Children

- **<expression usage="precondition">** or
  <variable usage="precondition"> or
  <collection usage="precondition"> or
  <script usage="precondition">
  Defines the precondition for the Log object.
- **<sequence usage="preprocessing">**
  Defines the preprocessing sequence for the Log object.
- **+<logItem>**
  Defines the list of Log items.
- **<eventHandling>**
  Defines the event handling for the Log object.
- **<tuning>**
  Defines the tuning settings for the Log object.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the Log object.

Example

```
<log>
  <logItem destination="file">
    <text>Manufacturer = </text>
    <variable link="#Car manufacturer"/>
    <text> / Model = </text>
    <variable link="#Car model"/>
  </logItem>
  <logItem destination="customDB">
    <text>insert into carrequests values (</text>
    <variable link="#Car manufacturer"/>
    <text>,</text>
    <variable link="#Car model"/>
    <text>)</text>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
  </logItem>
</log>
```

LogItem

Attributes

- **`label`**
  A text string providing a name for the Log item.
- **`layer`**
  Defines the layer for the Log item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **`category`**
  Defines the category for the Log Item. Legal values are *info*, *debug*, *error*, *statistics*, *billing*. If not specified, defaults to *info*.
- **`occurrence`**
  Defines the occurrence level for the Log item. Can be *always*, *once*, or an integer between 1 and 10. If not specified, defaults to *always*.
- **`destination`**
  Defines the logging destination for the Log Item. One of *default*, *mediaPlatform*, *customDB*, *file*, *logOSDM*, *systemDB*. If not specified, defaults to *default*.
  The destination *logOSDM* is only available when the OSDM integration is licensed.

Children

- **+(`<output>`, `<variable>`, `<expression>`, `<collection>`, `<script>`, `<text>`, `<layer>`)**
  Defines the string to be logged, as a concatenation of various elements.
  Note that a single blank will automatically be inserted between any two consecutive non-text children during import into VoiceObjects Desktop or deployment to VoiceObjects Server in order to ensure proper output behavior.

Example

```
<logItem category=”statistics”>
    <text>Manufacturer = </text>
    <variable link=”#Car manufacturer”/>
    <text> / Model = </text>
    <variable link=”#Car model”/>
</logItem>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The following table contains all object types that can reference a Log object:

Icon	Object Name	Use Case Example
	Module	A Log object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, a Log object is used within a Sequence object in a dialog flow.
	Menu	A Log object can be referenced within each Menu item in a Menu object.
	If	A Log object can be used in either the THEN item or the ELSE item of an If object.
	Case	A Log object can be used in any WHEN item of a Case object.
	Loop	A Log object can be used within the embedded sequence of a Loop object.
	Goto	A Log object can be referenced via a Goto object.
	Hyperlink	A Log object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Log object should indicate which type of information is logged. The table below lists three examples:

Name	Description
 Log car make and model	Logs the car make and model for which the caller wants to get an insurance quote
 Store billing record	Stores call data into a billing record via the `<log>` tag
 Update caller profile	Augments the caller's profile to include the interactions in the current call

Copyright © 2001-2011 Voxeo Germany GmbH

Loop

Overview

The Loop object allows you to repeat the processing of a sequence of objects while a given condition evaluates to *true*. The Loop object is finished when this condition evaluates to *false*.

It is possible to have VoiceObjects Server check the condition either before or after processing the given objects. This allows you to implement standard loop constructs of the type WHILE (condition) {…} and DO {…} WHILE (condition) known from other programming or scripting languages.

The Object Definition below covers the configuration of the Loop object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Loop object belongs to the object category Logic.

Dialog Flow Scenario

The following dialog flow example uses the Loop object to repeat the output of a confirmation number as long as the caller wishes to hear it.

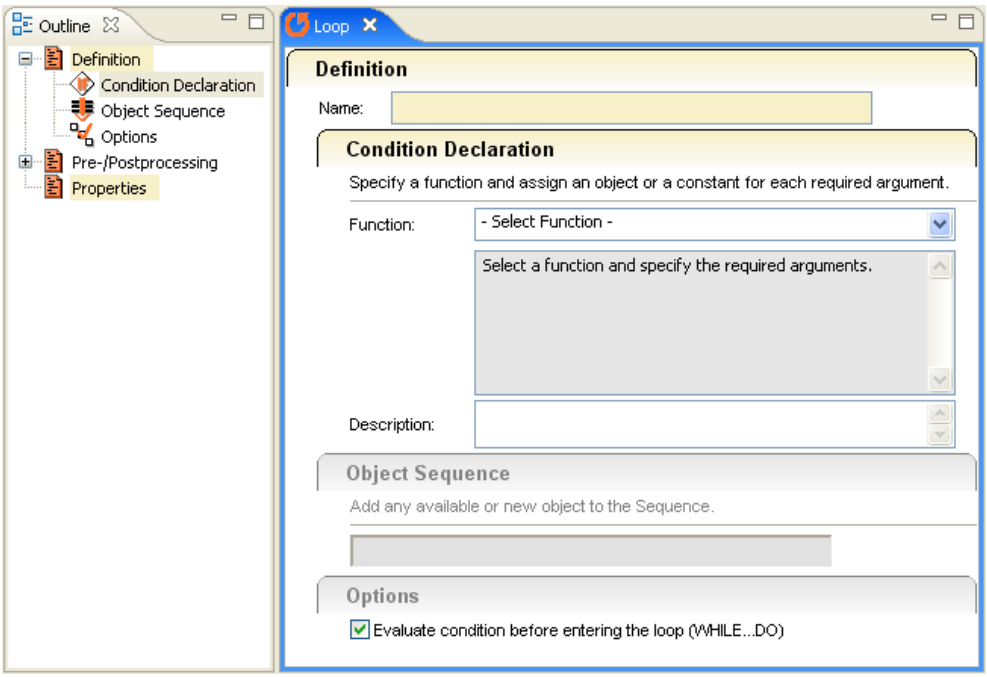
Object – Caller		Dialog Flow
		*Your confirmation number is 5 4 1 7 6.*
		*Shall I repeat the number?*
	 Caller	*Yes.*
		*Your confirmation number is 5 4 1 7 6.*
		*Shall I read out the number again?*
	 Caller	*No.*

Object Definition

The **Definition** of the Loop object provides the following sections:

- Condition Declaration
  To define the conditions for entering and continuing the loop.
- Object Sequence
  To specify the sequence of objects to be processed within each loop iteration.
- Options
  To specify processing options.

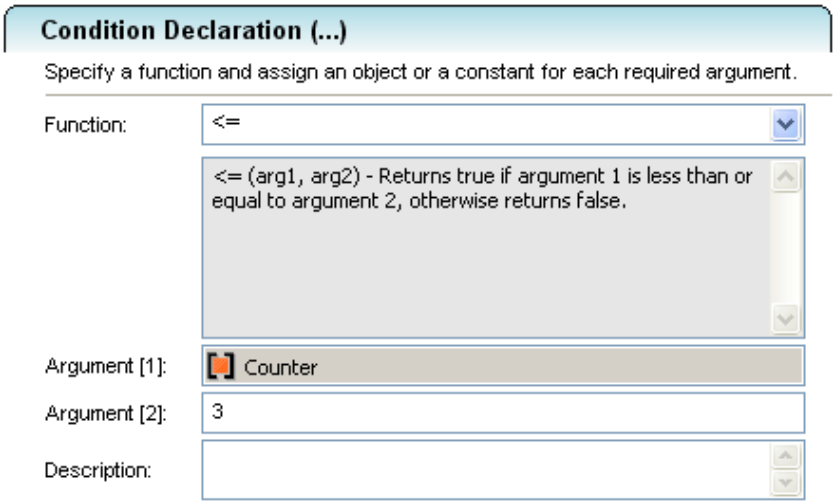
Copyright © 2001-2011 Voxeo Germany GmbH



For information regarding additional object configuration refer to *Precondition* and *Properties* in this *Object Reference*.

Condition Declaration

The **Condition Declaration** section defines the condition that needs to evaluate to *true* in order for the Loop object to continue. Either define an embedded condition by selecting an appropriate operator from the function drop-down list and specifying the required arguments, or link a Variable, Expression, Collection, Script, or Layer object that returns a Boolean value.

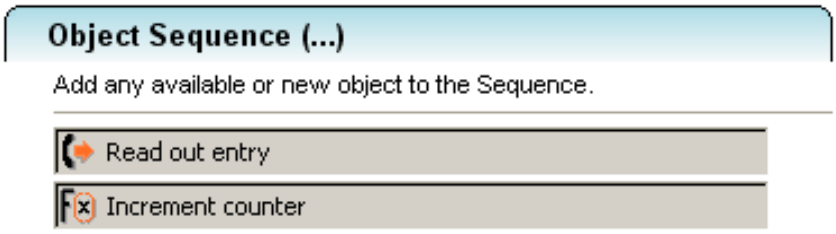


For more information on the use of functions, see the Expression object in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Object Sequence

The **Object Sequence** section contains the sequence of objects to be processed within each loop iteration. Instead of defining an embedded sequence, an autonomous Sequence object may be linked.

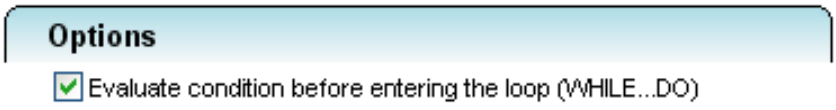


Options

The **Options** section defines whether the loop has a While behavior, or a Do-While behavior.

If the check box **Evaluate condition before entering the loop (WHILE…DO)** is selected (which is the default), the condition specified in the condition declaration is checked before first entering the object sequence. If it evaluates to *true*, the object sequence is entered. Otherwise, the Loop object ends without processing any object.

If the check box **Evaluate condition before entering the loop (WHILE…DO)** is clear, the condition declaration is first tested after the object sequence has been processed once. Only if it evaluates to true the object sequence is re-entered.



VoiceObjectsXML Definition

The Loop object is represented by the VoiceObjectsXML element `<loop>`. It has one attribute and five groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

Loop

Attributes

- **`prechecked`**
  Indicates the loop type. Can either be *true* or *false*. If not specified it defaults to *true*.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Loop object.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<sequence usage="preprocessing">**
  Defines the preprocessing sequence for the Loop object.
- **<expression usage="condition">** or
  <variable usage="condition"> or
  <collection usage="condition"> or
  <script usage="condition">
  Defines the loop condition (i.e. the condition under which the loop is continued).
- **<sequence>**
  Defines the sequence of objects to be processed within the loop.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the Loop object.

Example

```
<loop prechecked="false">
  <expression usage="condition" link="#LoopCondition"/>
  <sequence link="#Main loop sequence"/>
</loop>
```

Object Interoperability

The following table contains all object types that can reference a Loop object:

Icon	Object Name	Use Case Example
	Module	A Loop object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, a Loop object is used within a Sequence object in a dialog flow.
	Menu	A Loop object can be used as the destination object within a Menu item inside a Menu object.
	Confirmation	A Loop object can be used as the destination object within a Correction item inside a Confirmation object.
	If	A Loop object can be either the THEN item or the ELSE item of an If object.
	Case	A Loop object can be used in any WHEN item of a Case object.
	Loop	A Loop object can be used within the embedded sequence of another Loop object.
	Goto	A Loop object can be referenced via a Goto object.
	Hyperlink	A Loop object can be the destination of a Hyperlink object.

Copyright © 2001-2011 Voxeo Germany GmbH

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Loop object should describe what is done within the loop. The short description should contain a brief explanation of the loop condition, and whether the loop is pre- or post-checked. The table below lists two examples:

Name	Description
Repeat confirmation number	Plays the confirmation number at least once and repeats the number as long as the caller wishes to hear it.
Retrieve desired back-end info	Retrieves data from the back-end for all desired items. Loop is entered only if at least one item is selected, and loops until all items have been retrieved from the back-end.

Copyright © 2001-2011 Voxeo Germany GmbH

Menu

Overview

The Menu object presents a list of choices and lets the caller select one of them. When the caller has selected an item, the dialog continues with the processing of the object referenced in that item. After the subdialog is finished, the dialog flow may optionally return to the menu.

The Object Definition below covers the configuration of the Menu object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Menu object belongs to the object category Components.

Dialog Flow Scenario

The following dialog flow example shows a simple menu dialog within an information portal. The caller selects *Stock Quotes* from a list of available Menu items and after processing the *Stock Quotes* subdialog the dialog flow returns to the main menu.

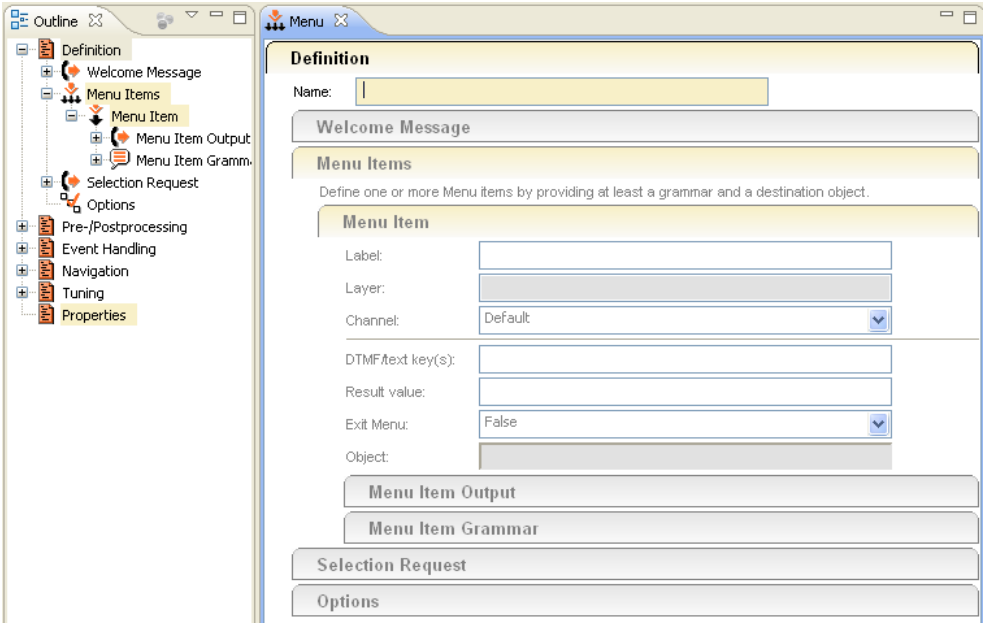
Object – Caller	Dialog Flow
	*Main menu. You can choose from among the following services: News, Stock Quotes, Travel, and Entertainment. Which are you interested in?*
	*Stock Quotes*
	**[Process dialog flow of:]** *Stock Quotes*
	**[Repeat menu:]** *Back in the main menu. Choose again: News, Stock Quotes, Travel, or Entertainment.*

Object Definition

The **Definition** of the Menu object provides the following sections:

- Welcome Message
  To define the introductory output for the entire menu.
- Menu Items
  To define the list of individual menu choices available.
- Selection Request
  To define the output playing a final selection request message.
- Options
  To configure processing options.

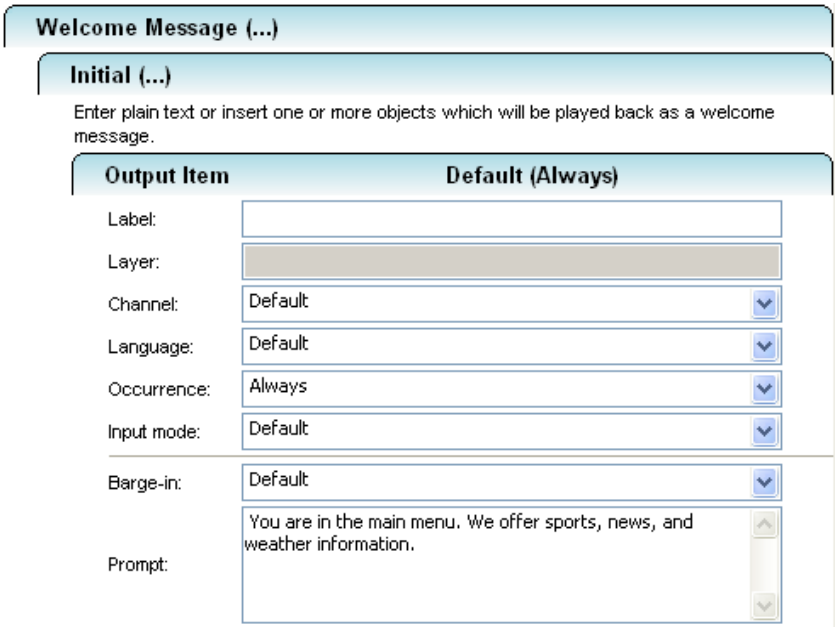
Copyright © 2001-2011 Voxeo Germany GmbH



For information regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Navigation*, *Tuning*, and *Properties* in this *Object Reference*.

Welcome Message

The **Welcome Message** defines an optional output to be presented to the caller in the **Initial** section, as well as modified outputs for event cases in the **Event Reprompt** section (only processed in the voice and video channel).



In the voice and video channel, the optional welcome message in the **Initial** section may contain the list of Menu items available. Alternatively, this list may be provided by the Menu items themselves (see below). If Menu items are dynamically added or removed via custom layers (see Chapter 7 – *How to Use Layers* in the *Design Guide*),

Copyright © 2001-2011 Voxeo Germany GmbH

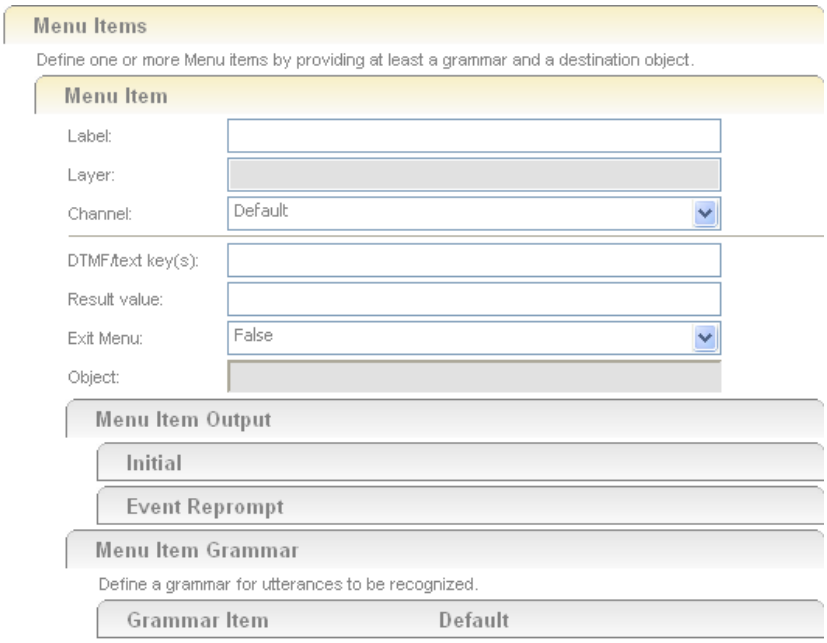
their individual outputs should be used.
In the text and Web channel, the outputs of the Menu items must be used to present the items. In these channels, the welcome message only holds the introductory text, e.g. *Main menu.|Please choose:*.

For further information on the usage of the pipe symbol "|" in the text and Web channel, and in general for details on how to configure an Output object, refer to Output in this *Object Reference*.

For a detailed description of how to use the reprompting mechanism provided in the **Event Reprompt** section, refer to Chapter 8 – *Advanced Event Handling* in the *Design Guide*. Note that reprompting only applies to the voice and video channel.

Menu Items

In the **Menu Items** section, one or more Menu items can be defined.



In each **Menu Item** section the following properties can be specified:

Property	Description
Label	Optional parameter to identify the Menu item in a list.
Layer	Optional parameter to indicate the layer that the Menu item belongs to. For more information on layers, see *Layers* in this *Object Reference* or Chapter 7 – *How to Use Layers* in the *Design Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Channel	Identifies the channels in which the respective Menu item is to be active. If a Menu item is appropriate for all channels, the default channel layer setting *Default* should be used. Otherwise, define one of the four channels *Voice*, *Video*, *Text* or *Web*. For your convenience, use the two combined settings *Voice/Video* or *Text/Web* in case the Menu item is to be active in those combinations. Using a combined channel is the same as defining two Menu items with the individual channel setting, with all other settings being the same.
Object	Specifies the object that is to be processed when the caller selects this Menu item.
DTMF/text key(s)	Allows defining keys (DTMF keys in the voice and video channel, and access keys in the text channel) that can activate this Menu item. Typically, exactly one key or key sequence is associated with an item, but it is possible to allow more than one key or key sequence to activate a Menu item. In that case, define a comma-separated list of keys. Examples: • "1" Responding with "1" can activate the current Menu item. • "3, 10" Responding with "3" or "10" can activate the current Menu item. Note that in the voice and video channel, the tuning property *Input - DTMF Inter-Digit Timeout* will control the timeout allowed between two key strokes. ⓘ ▸▸ **Note:** If DTMF/text keys are defined for an item, any corresponding grammar definition specified in the embedded Grammar object of the item will be ignored (i.e. DTMF grammars in the voice and video channel, and voice grammars in the text and Web channel). Furthermore, if auto-numbering (see Options) is active, any settings in the **DTMF/text key(s)** field and again the corresponding grammars will be ignored.
Result value	Optional value that is assigned to the **Result variable** defined in the Options section when the current Menu item is selected. The **Result value** property can dynamically be set at call time by defining it with a Variable, Expression, Layer, or Script object.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Exit Menu	Indicates what to do after the object defined for this Menu item has been processed. The default setting *False* indicates that the dialog returns back to the menu. When set to *True*, the processing of the Menu object is terminated and the dialog continues with the next object after the Menu object (if any). The **Exit Menu** property can dynamically be set at call time by defining it with a Variable, Expression, or Script object. If it is dynamically set, the value is determined before the object defined for the Menu item is processed.

In addition, each **Menu Item** section contains the following two subsections:

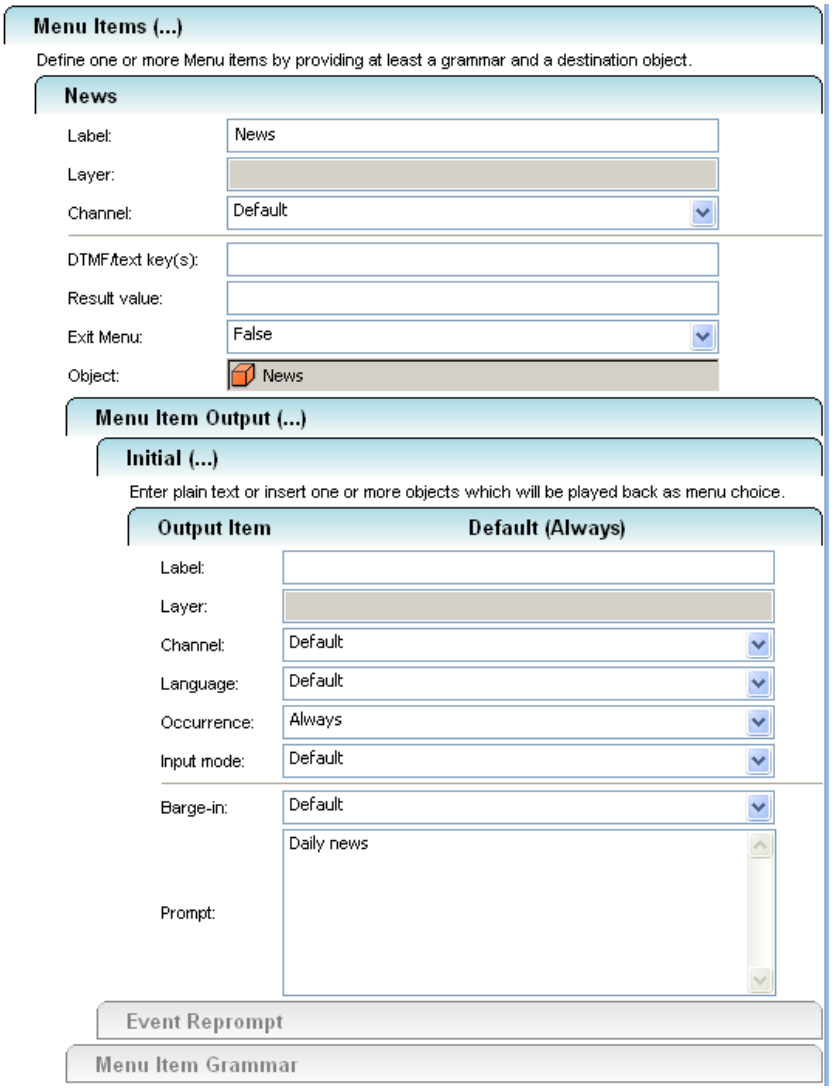
- Menu Item Output
- Menu Item Grammar

Menu Item Output

In the **Menu Item Output** section, output can be specified that presents this specific Menu item to the caller.
In the voice and video channel you can either decide to include information on the items in the welcome message or to use the individual Menu item outputs. Choose the first option if the menu and its items do not change dynamically during or between calls. Choose the second option if some Menu items are associated with custom layers, so that they could be hidden for some callers.
In the text and Web channel you must use the Menu item outputs to present the items. The welcome message is here only used to hold the introductory text, e.g. *Main menu.*|*Please choose:*.

In all channels, the outputs of all Menu items are presented in the order the items are defined in, the welcome message coming first. In the Web channel, the tuning parameter *Presentation – Menu* governs how Menu items are to be arranged on the screen. Possible values are *horizontal* (the default; all items are displayed in one row), *vertical* (all items are displayed one below the other), *2columns* (two items are displayed in one row), *3columns* (three items are displayed in one row).

Copyright © 2001-2011 Voxeo Germany GmbH



For further details on how to configure an Output object refer to Output in this *Object Reference*.

For a detailed description of how to use the reprompting mechanism provided on the **Event Reprompt** section, refer to Chapter 8 – *Advanced Event Handling* in the *Design Guide*. Note that reprompting only applies to the voice and video channel.

Menu Item Grammar

The **Menu Item Grammar** defines the voice/text and DTMF input that invokes the processing of this Menu item. In the voice or video channel, the grammar can be left empty, in which case the server automatically generates a voice grammar from the name of the object specified as the Menu item object. For better control over the utterances that activate the Menu item, it is highly recommended to define a proper grammar, though.

In the text channel, the grammar is used to define the access key that activates the Menu item. It will be displayed on the screen, together with the Menu item output. The grammar must be defined in the **Voice/Text** section of the Grammar object.

Copyright © 2001-2011 Voxeo Germany GmbH

A typical display of a Menu object in the text channel could look like this:



In this example, the Menu item grammars define the access keys (1, 2, 3, 4), whereas the Menu item outputs define the option names (*Service plans*, *Customer data*, etc.).

In the Web channel, menu items will be displayed as clickable hyperlinks. No access keys will be shown, so the Menu item grammar isn't conceptually required in this channel. To make the Menu object valid, though, either define ? (question mark) as the Menu item grammar, or enable auto-numbering in the **Options** section.

For further details on how to specify a Grammar object refer to Grammar in this *Object Reference*.

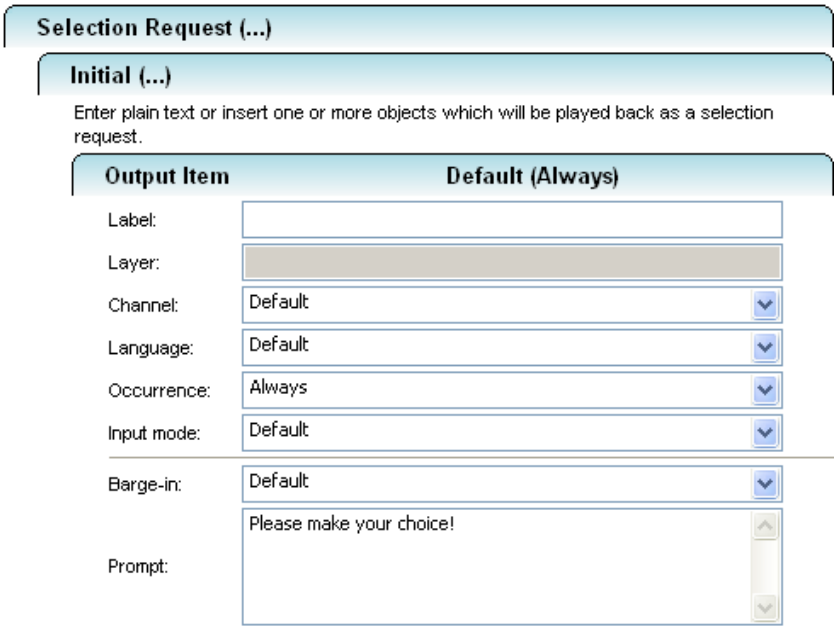
Note: In the voice and video channel, a DTMF grammar for an item should not be defined in the **DTMF** section of the **Menu Item Grammar**, but in the **DTMF/text key(s)** field. With the latter approach, DTMF grammars will be rendered into VoiceXML using the "dtmf" attribute on the `<choice>` element. If you do not define DTMF/text key(s) but a DTMF grammar instead, it will be rendered using a `<grammar>` element as a child of `<choice>`, which is not supported on all media platforms.
If no DTMF grammar is defined in the Menu items and **Auto-numbering** is disabled, an *Error – Internal* event is activated.

Tip: A Menu object may contain any number of Menu items. However, in the case of voice applications, it is recommended to limit this to a number that is easy to remember for callers, typically not more than 5 items.

Selection Request

The **Selection Request** defines an optional output to be presented to the caller in the **Initial** section, as well as modified outputs for event cases in the **Event Reprompt** section (only processed in the voice and video channel).

Copyright © 2001-2011 Voxeo Germany GmbH



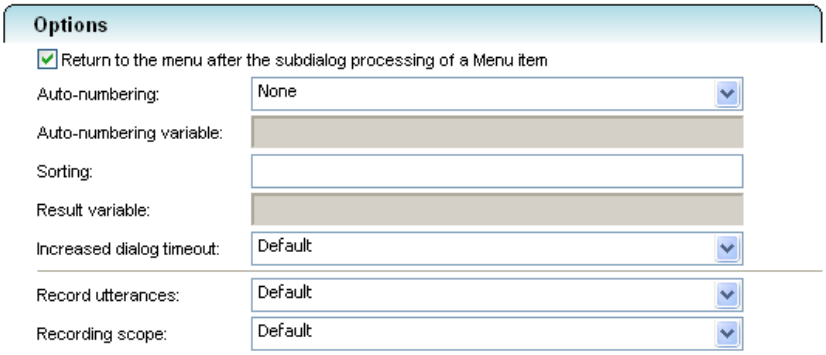
In the voice and video channel, the optional selection request in the **Initial** section may contain a request to choose one of the existing Menu items, e.g. *Please make your choice now*. This is especially helpful if Menu items are dynamically added or removed using custom layers (see Chapter 7 – *How to Use Layers* in the *Design Guide*).

For further information on the usage of the pipe symbol "|" in the text and Web channel, and in general for details on how to configure an Output object, refer to Output in this *Object Reference*.

For a detailed description of how to use the reprompting mechanism provided in the **Event Reprompt** section, refer to Chapter 8 – *Advanced Event Handling* in the *Design Guide*. Note that reprompting only applies to the voice and video channel.

Options

There are four processing options for the Menu object.



Return to the menu after the subdialog processing of a Menu item defines how to continue in the dialog after the processing of a Menu item is finished. If this check box is selected (the default), the processing of the menu only ends if a Menu item is

Copyright © 2001-2011 Voxeo Germany GmbH

processed with **Exit Menu** set to *True*. Otherwise the processing of the Menu object continues. If this check box is clear, the processing of the Menu object ends after processing any Menu item regardless of the respective setting in the **Exit Menu** field. This option is indicated in the **Dialog Designer** of VoiceObjects Desktop by the  icon next to the Menu object icon.

Auto-numbering is a feature that automatically assigns keys to each consecutive Menu item, starting with 1. These keys can be used to activate a Menu item, using DTMF in the voice and video channel, or typing in the number as a response in the text channel. In the Web channel, auto-numbering is not relevant, as the keys will not be displayed.
If auto-numbering is active, any DTMF keys or DTMF grammars defined in any item will be ignored in the voice and video channel, and any text keys or voice/text grammars in the text channel. If you do not wish to use auto-numbering, leave the setting at *None*. There are two different types of auto-numbering:
1.) *Continuous*
Continuous auto-numbering means that first all layers associated with Menu items are evaluated, so that items might get disabled, and then keys are assigned. This will always result in a continuous assignment from 1 to n, where n is the remaining number of items after the layer evaluation.
As an example, consider a Menu object with 5 items, where the second one has a layer that evaluates to *false* for a given call. This second item is first ruled out, and then all remaining items get their numbers. Thus, the first one can be activated with "1", the second one with "2", the third one with "3" and the fourth one with "4".
2.) *Discontinuous*
Discontinuous auto-numbering means that first all Menu items will get a key assigned, and then the layer evaluation takes place. This might result in a discontinuous assignment, meaning that there can be items that retain their key but are ruled out due to layer conditions. Discontinuous auto-numbering allows keeping key assignments to Menu items that the caller might be familiar with, while at the same time using layers to rule out certain Menu items.
As an example, consider a Menu object with 5 items, where the second one has a layer that evaluates to *false* for a given call. First, all items get their numbers, i.e. the first item will get "1", the second "2", etc. Due to the layer of the second item, there are four remaining Menu items; the first can be activated with "1", the second one with "3", the third one with "4" and the fourth one with "5".

Auto-numbering variable can be used to define a reference to a Variable object that receives the DTMF key assigned to each Menu item in auto-numbering, e.g. to read out the key within the respective Menu Item Output. Note that the value is provided within the scope of the respective Menu Item Output, and that it may be undefined outside of this scope.

Sorting enables the dynamic re-sorting of Menu items at call time. It can be set at call time by defining it with a Variable, Expression, or Script object. The value provided needs to be the comma-separated list of Menu item labels. Note that all Menu items need to have labels defined for this to work properly. Any Menu items that do not occur within the sorting string are filtered out.
Sorting is always performed first (if the **Sorting** property is defined); afterwards filtering and auto-numbering are applied as described above under **Auto-numbering**.

Result variable defines a reference to a Variable object that receives the result value defined for the selected Menu item. This value can then be used within the subdialog that is initiated by the Menu item selection.

Increased Dialog Timeout specifies a timeout value that is applied for this object in case it is higher than the current dialog timeout in a session. If it is lower, the setting is ignored. It is mainly applicable in the text and Web channel, where the dialog timeout of a service should be set to a relatively low value, e.g. 40 seconds, to keep the number of concurrent sessions low, as they usually run into the server timeout due to

Copyright © 2001-2011 Voxeo Germany GmbH

technical reasons. Use this setting to increase the timeout temporarily for objects that are likely to require more time to respond to, e.g. when the caller has to type in an address.

Record utterances specifies whether utterances made in this object are to be recorded (*True*) or not (*False*), given that utterance recording is switched on for this session. *Always* indicates that utterances in this object are always recorded when utterance recording is enabled on the service, regardless of statistical filtering. By default the value is *Default*, indicating that the current value of the corresponding dialog context setting RECORDUTTERANCES is used.

Recording scope defines the scope in which recordings will be made. If set to *Recognition*, only those utterances will be recorded that were successfully recognized (including the activation of hyperlinks). If set to *No Match*, only those utterances will be recorded that triggered a No Match event. If set to *All*, any recording will be stored. If set to *Default*, the current value of the corresponding dialog context setting RECORDINGSCOPE is used. By default the value is *Default*.

Due to the nature of recordings, the feature of recording utterances is only available in the voice and video channel. For more information on this refer to Chapter 6 – *Recording of Utterances* in the *Deployment Guide*.

VoiceObjectsXML Definition

The Menu object is represented by the VoiceObjectsXML element `<menu>`. It has eight attributes and eleven groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<menu>` element uses the embedded `<menuItem>` element.

Menu

Attributes

- **`return`**
  Defines whether the dialog returns to the Menu object after a Menu item has been activated. Either *true* or *false*. If not specified, defaults to *true*.
- **`autonumbering`**
  Defines whether automatic numbering for the Menu items should be used. Can be *none*, *continuous*, or *discontinuous*. If not specified, defaults to *none*.
- **`autonumberingVariable`**
  Defines a reference to a Variable object that provides the key assigned to each Menu item.
- **`sorting`**
  Defines a sorting string for the Menu items. Can be a static value or a reference to a Variable, Expression, or Script object.
- **`resultVariable`**
  Defines a reference to a Variable object that provides the result value for the selected Menu item.
- **`recordUtterances`**
  Defines whether to use utterance recording. One of *true*, *false*, *always*, *default*. If not specified, defaults to *default*.
- **`recordingScope`**
  Defines the scope for utterance recording. One of *all*, *noMatch*, *recognition*, *default*. If not specified, defaults to *default*.

Copyright © 2001-2011 Voxeo Germany GmbH

- **increasedDialogTimeout**
  Defines the increased dialog timeout in seconds. If not specified, the setting from the dialog context is applied at call time.

Children

- **<expression usage="precondition">** or
  <variable usage="precondition"> or
  <collection usage="precondition"> or
  <script usage="precondition">
  Defines the precondition for the Menu object.
- **<sequence usage="preprocessing">**
  Defines the preprocessing sequence for the Menu object.
- **<output type="initial">**
  Defines the "initial" section of the menu welcome message.
- **<output type="reprompt">**
  Defines the "event reprompt" section of the menu welcome message.
- **+<menuItem>**
  Defines the list of Menu items.
- **<output usage="final" type="initial">**
  Defines the "initial" section of the menu selection request.
- **<output usage="final" type="reprompt">**
  Defines the "event reprompt" section of the menu selection request.
- **<eventHandling>**
  Defines the event handling for the Menu object.
- **<customNavigation>**
  Defines the navigation settings for the Menu object.
- **<tuning>**
  Defines the tuning settings for the Menu object.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the Menu object.

Example

```
<menu dtmf="continuous">
  <output>
    <outputItem><text>This is the main menu. Press one to file a new
    order. Press two to check on an existing
    order.</text></outputItem>
  </output>
  <menuItem label="NewOrder">
    <module link="#File new order"/>
  </menuItem>
  <menuItem label="ExistingOrder">
    <module link="#Check existing order"/>
  </menuItem>
  <output usage="final">
    <outputItem><text>Please make your choice now.</text></outputItem>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
  </output>
</menu>
```

MenuItem

Attributes

- **`label`**
  A text string providing a name for the Menu item.
- **`layer`**
  Defines the layer for the Menu item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **`exit`**
  Defines whether the dialog exits the Menu object after processing this Menu item. Either *true* or *false*. If not specified, defaults to *false*.
- **`resultValue`**
  Defines the result value to be set when this Menu item is selected. Can be a static value or a reference to a Variable, Expression, Layer, or Script object.
- **`dtmfKeys`**
  Defines DTMF/text keys to be associated with the corresponding Menu item. Can be one key or key sequence, or a comma-separated list of keys or key sequences.

Children

- **`<output usage=”menu” type=”initial”>`**
  Defines the “initial” section of the Menu item output.
- **`<output usage=”menu” type=”reprompt”>`**
  Defines the “event reprompt” section of the Menu item output.
- **`<grammar>`**
  Defines the activation grammar for the Menu item. Optional if auto-numbering is enabled in the Menu object. Instead of defining DTMF grammars here, use the `dtmfKeys` attribute on the `<menuItem>` element.
- One of the following:
  `<module>`, `<input>`, `<output>`, `<sequence>`, `<menu>`, `<confirmation>`, `<list>`, `<osdm>`, `<plugin>`, `<connector>`, `<script>`, `<log>`, `<if>`, `<case>`, `<goto>`, `<loop>`, `<expression>`, `<recording>`, `<transfer>`, `<pause>`, `<exit>`
  Defines the object to be processed when the Menu item is selected.

Example

```
<menuItem label=”1” exit=”true”>
  <output usage=”menu”>
    <outputItem><text>Item one.</text></outputItem>
  </output>
  <output>
    <outputItem><text>You have selected Menu item
one.</text></outputItem>
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
    </output>
</menuItem>
```

Object Interoperability

The following table contains all object types that can reference a Menu object:

Icon	Object Name	Use Case Example
	Module	A Menu object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, a Menu object is used within a Sequence object in a dialog flow.
	Menu	A Menu object can be used as the destination object within a Menu item.
	Confirmation	A Menu object can be used as the destination object within a Correction item inside a Confirmation object.
	If	A Menu object can be either in the THEN or the ELSE item of an If object.
	Case	A Menu object can be used in any WHEN item of a Case object.
	Loop	A Menu object can be used within the embedded sequence of a Loop object.
	Goto	A Menu object can be referenced via a Goto object.
	Hyperlink	A Menu object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Menu object should identify the type of menu that is presented to the caller. The short description should contain a brief explanation of the choices that are offered. The table below lists three examples:

Name	Description
 Main menu	Main menu of the Entertainment Portal, offering the choice between Joke box, Quiz, Blackjack, and Chat.
 Account menu	Determines which type of account the caller wants to access: checking, savings, or money market.
 Card Services menu	Offers the caller the available card services: Sign up for a new card, cancel an existing card, get a partner card, or report a lost/stolen card. Gold members additionally have the option of travel booking.

Copyright © 2001-2011 Voxeo Germany GmbH

Module

Overview

The Module object typically represents the start node in a dialog flow and is used to group objects into logical units.

A Module object is intended to represent a standalone or subdialog, such as logging into an account or performing a money transfer in a banking application. Structuring dialogs into logical units helps designers keep a clear and easy-to-follow dialog structure – the display of a dialog flow in the **Dialog Designer** of VoiceObjects Desktop stops at Module objects.
Furthermore, the Module object allows rapidly developing new applications based on existing building blocks, due to the reusability of complete modules.

In addition, the Module object allows you to provide global settings for event handling, navigation, tuning, as well as OSDM parameters, which can be inherited by subordinate objects in the dialog flow. Based on the object-oriented approach, global settings defined in a Module object can be overwritten locally within the subordinate objects.

For more information on reusability and inheritance, see Chapter 1 – *Application Design - The Modular Concept* in the *Design Guide*.

With Infostore and VoiceObjects Analyzer, modules can be used to store and retrieve statistical information on how the caller controls and navigates within an application. As such, modules group parts of the application into logical units that can later be analyzed in terms of event frequency, duration of stay, etc. There are several predefined reports related to Module objects.

The integrated dialog flow logic of the Module object is to present an optional title output and an optional welcome message, to process the embedded sequence of referenced objects and finally to present an optional goodbye message before exiting.

The Object Definition below covers the configuration of the Module object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Module object belongs to the object category Components.

Dialog Flow Scenario

The following dialog flow example describes a simple dialog in a flight booking application.

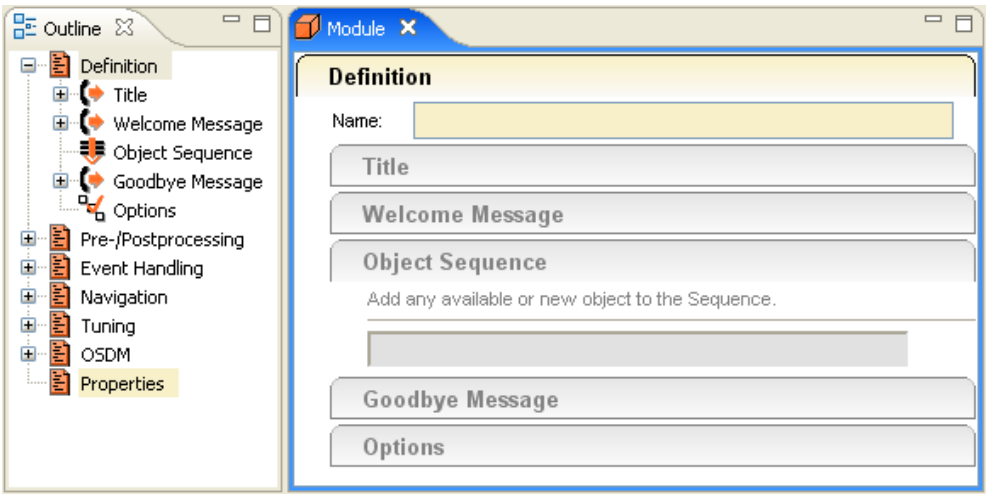
Object – Caller	Dialog Flow
	*Welcome to World Travel Airline.*
	**Process dialog flow of:** [ Flight Booking Menu]
	*Thanks for calling. Goodbye.*

Copyright © 2001-2011 Voxeo Germany GmbH

Object Definition

The **Definition** of the Module object consists of the following sections:

- Title
  To define an optional title that is repeated in every subordinate object.
- Welcome Message
  To play an optional welcome message to the caller.
- Object Sequence
  To specify a sequence of objects to be processed.
- Goodbye Message
  To provide an optional goodbye message.
- Options
  To configure processing options.



The **Module** editor provides dialog end processing and a reference library within the **Pre-/Postprocessing** section, which are both explained below.

In addition, it contains an **OSDM** section, which is also explained in this document.

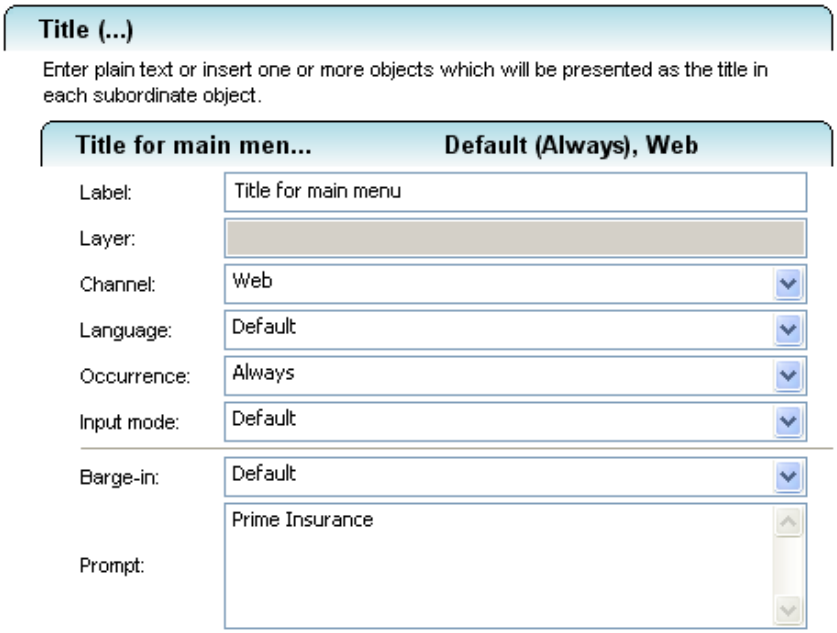
For further information regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Navigation*, *Tuning*, or *Properties* in this *Object Reference*.

Title

In the **Title** section (represented as a tab on the **Welcome Message** section in Desktop for Web), an optional output can be specified, that is repeated in each child object of the module (plus the Module object itself, i.e. for the welcome and goodbye message). In the voice, video and text channel, it is played or displayed before the *main* prompt of each object, i.e. before the input request of an Input object, before the welcome message of a Menu object, before the confirmation request of a Confirmation object, etc. It is not presented in outputs that are defined as the object to be processed in event handling or navigation items. Note that the main output of an object directly follows the title output, with no spaces or line breaks added. To define a line break in the text channel, use the pipe symbol "|".

Copyright © 2001-2011 Voxeo Germany GmbH

In the Web channel, the title is rendered as the page's title, which means that the Web browser typically shows it as a kind of header at the very top of the screen, separated from the body content by some special layout.



In the voice and video channel, the title output could be a "land marker" earcon, to notify the caller about the progressing dialog flow. In the text and Web channel, the title output could be a header that is repeated for each dialog step to let the callers know where in the flow they currently are. It is mainly used in the Web channel, as the text channel has tight length restrictions.

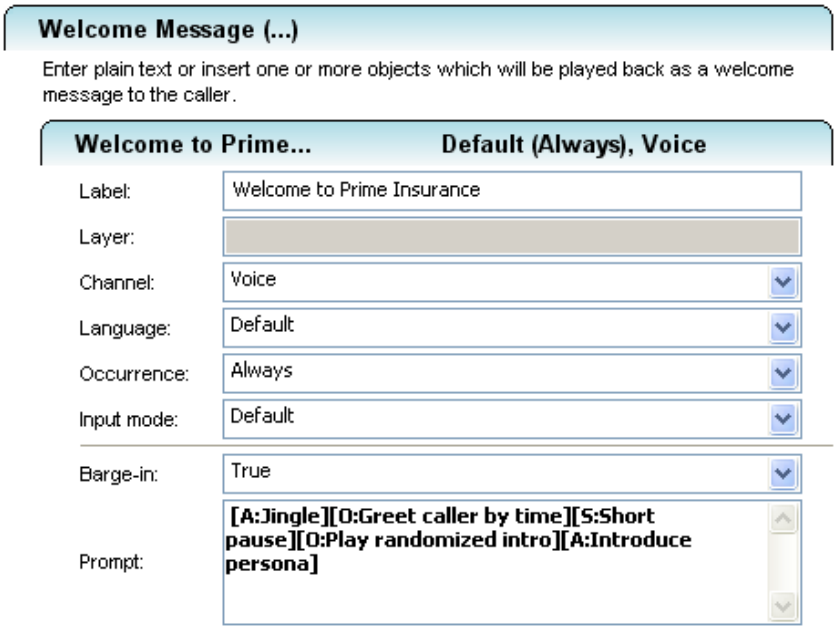
In all channels, the title output is only presented in the scope of the current Module object. Example: if a Module object with a title definition has another module without a title definition as a child object, then all children of the second module will not have any title.

For further details on how to configure output refer to the Output object in this *Object Reference*.

Welcome Message

In the **Welcome Message** section (i.e. its **Welcome** section in Desktop for Web), an optional output can be specified. In the voice and video channel this is typically used to play an introduction to this subdialog. In the text and Web channel, the welcome message could be an introductory text. To move on to the next dialog step, the caller has to activate a "Proceed" button or option. Since this requires an extra interaction, the welcome message could be left empty in the text and Web channel.

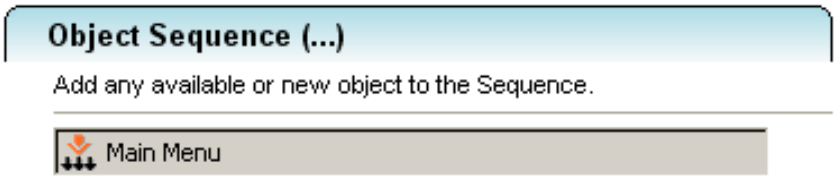
Copyright © 2001-2011 Voxeo Germany GmbH



For further details on how to configure output refer to the Output object in this *Object Reference*.

Object Sequence

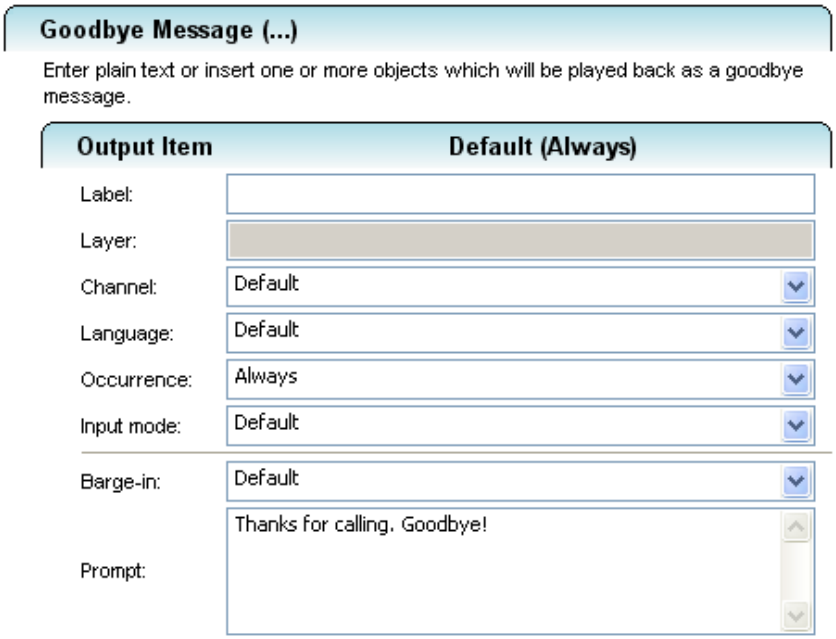
The **Object Sequence** references a sequence of objects to be processed after the welcome message has been presented. Instead of individually inserting objects into the embedded sequence, a Sequence object can be linked.



Goodbye Message

The **Goodbye Message** is an optional output that is presented before leaving the Module object. It is typically used to play or display an exit output for the subdialog.

Copyright © 2001-2011 Voxeo Germany GmbH



For further details on how to configure output refer to the Output object in this *Object Reference*.

Options

The **Options** section contains one option that specifies if the Module object is included in the module history list maintained by VoiceObjects Server. By default, this option is enabled (i.e. check box selected).

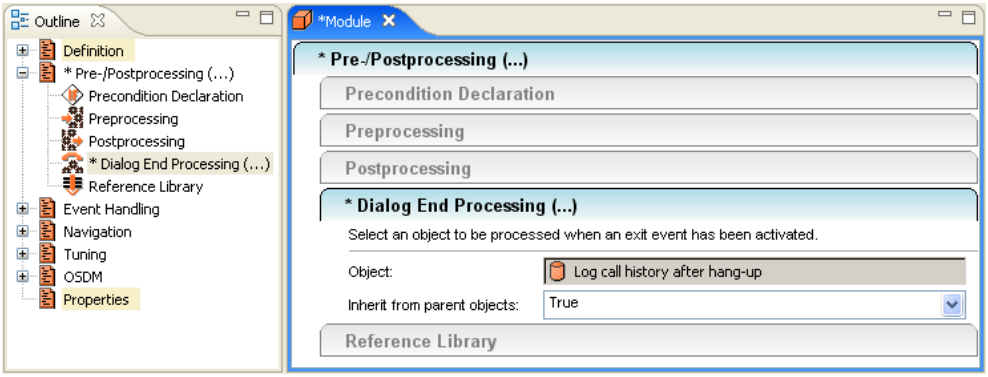


The module history can be retrieved by using the Expression functions MODULEHISTORY() and MODULESET(). It is useful for both in-dialog and post-dialog statistics.

Dialog end processing

Dialog end processing (specified in the **Pre-/Postprocessing** section of the **Module** editor) offers the capability to perform certain actions after a dialog has ended. Typically, these actions make use of a Connector object e.g. to clean up an abandoned shopping cart or to write data to a CRM (Customer Relationship Management) system.

Copyright © 2001-2011 Voxeo Germany GmbH



In the **Object** field the object is specified that is processed when the dialog end occurs. Often, this is a Connector or a Log object. The option **Inherit from parent objects** defines whether dialog end processing in parent modules is also processed (*True*, the default) or whether dialog end processing stops after the processing of the locally defined object (*False*). If inheritance is set to *True*, the processing of the objects starts with the innermost module (the local one) and iterates through all parent modules until either the outermost module is reached or inheritance is set to *false* in some module along the path.

Within the dialog end processing, the reason for the dialog end can be accessed via the system expression *System:Exit Event*. Possible values are:

Exit value	Description
None	The dialog has not ended yet. This is the return value when calling *System:Exit Event* during an ongoing dialog.
Timeout	The dialog terminated due to a dialog timeout.
Exception	The dialog terminated due to an exception in VoiceObjects Server.
EndOfDialog	The dialog terminated because it reached the end of the application dialog flow.
CallerExit	The dialog terminated because the caller hung up.
Transfer	The dialog terminated due to a blind transfer (see the Transfer object in this *Object Reference* for more details).

Note that since the dialog itself has already ended, there is no use in placing any interactive dialog elements such as Output objects, Input objects, etc. into the **Object** field in the **Dialog End Processing** definition. In fact, if you do use such an object within dialog end processing, the server will not generate the corresponding markup code for it but simply ignore it.

Note: A license for a session is only freed after dialog end processing has finished. The processing takes place asynchronously. If you make use of service chaining, i.e. the consecutive calling of different VoiceObjects services, it cannot be guaranteed that dialog end processing is finished before the next service starts.

To send custom markup code to the media platform in case of a caller-side end of the dialog (either via hang-up, or a transfer) a custom event handler may be defined. For more details, see the description of event handling in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

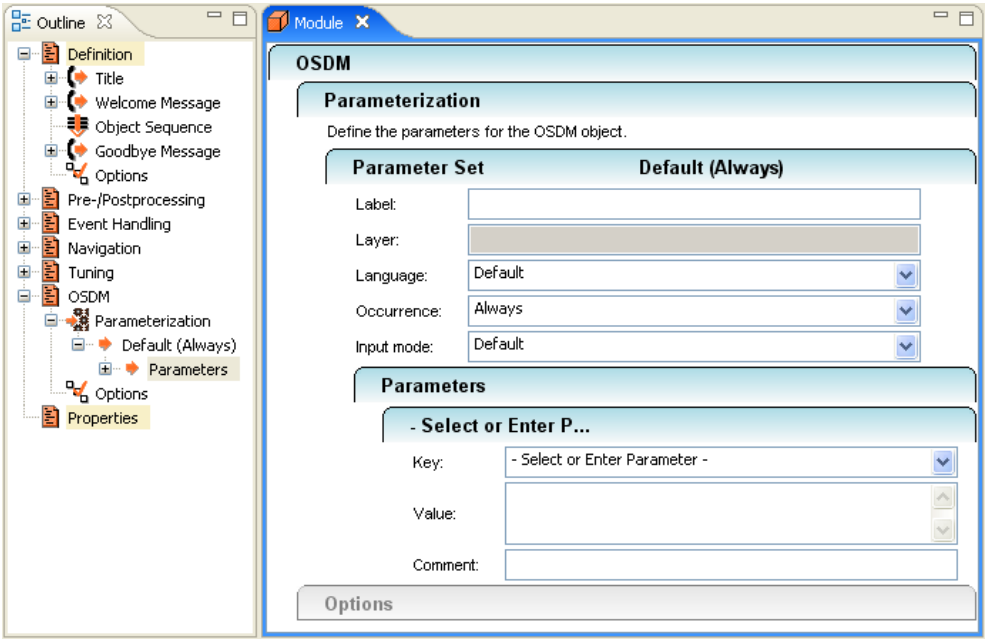
Reference library

The **Reference Library** section is found in the **Pre-/Postprocessing** section of the **Module** editor. It allows linking objects that are indirectly referenced within a Goto or List object but not used directly in the dialog tree.
The Goto object allows the processing of an object, the reference ID of which is dynamically evaluated at call time via a Variable object. In the List object, the reference ID of the object to be processed may come out of the data collection. In both cases, the required object does not need to be part of the dialog itself.
If an object is not referenced at all in an application, the server does not load it into its cache on deployment. Thus, trying to reference it during the dialog would lead to an error. The reference library is a means to link this kind of objects, in order to ensure that the server has them in its cache.

For further information on processing objects via reference IDs see the List object in this chapter or the Goto object in this *Object Reference*.

OSDM

If you make use of OSDM objects in your application, you can define general OSDM parameters in a Module object. They will always be inherited by subordinate OSDM objects. If you want to overwrite parameters in a subordinate OSDM object, you can do so by defining it there, using the same parameter name in the **Key** field as in the corresponding Module object. Inheritance can only be disabled in Module objects, by setting it to *False* in the **Options** section within the **OSDM** section.



For more information on how to define the OSDM parameters, refer to the OSDM object in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

VoiceObjectsXML Definition

The Module object is represented by the VoiceObjectsXML element `<module>`. It has three attributes and thirteen groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

Module

Attributes

- **`historyTracking`**
  Defines whether the Module object shows up in the module history. Either *true* or *false*. If not specified, defaults to *true*.
- **`dialogEndProcessing`**
  Reference to an object that is used for dialog end processing.
- **`inheritDEP`**
  Defines whether the dialog end processing of a higher-level Module object should be processed after the DEP of the current Module object was processed. Either *true* or *false*. If not specified, defaults to *true*.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Module object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Module object.
- **`<output usage="welcome">`**
  Defines the welcome message for the Module object.
- **`<output usage="title">`**
  Defines the title output for the Module object.
- **`<sequence>`**
  Defines the object sequence for the Module object.
- **`<output usage="goodbye">`**
  Defines the goodbye message for the Module object.
- **`<eventHandling>`**
  Defines the event handling for the Module object.
- **`<standardNavigation>`**
  Defines the standard navigation settings for the module.
- **`<customNavigation>`**
  Defines the custom navigation settings for the Module object.
- **`<tuning>`**
  Defines the tuning settings for the module.
- **`<sequence usage="postprocessing">`**
  Defines the postprocessing sequence for the Module object.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<sequence usage="library">**
  Defines the reference library for the Module object.
- **<osdmItems>**
  Defines the OSDM settings for this Module object.

Example

```
<module name="PrimeInsurance">
  <output usage="welcome" link="#intro"/>
  <sequence link="#Main sequence"/>
</module>
```

Object Interoperability

The following table contains all object types that can reference a Module object:

Icon	Object Name	Use Case Example
	Module	A Module object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, a Module object is used within a Sequence object in a dialog flow.
	Menu	A Module object can be used as the destination object within a Menu item.
	Confirmation	A Module object can be used as the destination object within a Correction item inside a Confirmation object.
	If	A Module object can be used either within the THEN item or the ELSE item of an If object.
	Case	A Module object can be used in any WHEN item or the ELSE item of a Case object.
	Loop	A Module object can be used within the embedded sequence of a Loop object.
	Goto	A Module object can be referenced via a Goto object.
	Hyperlink	A Module object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Module object should identify what the (sub)dialog represented by the Module object does. The short description should contain a brief explanation of assumptions and restrictions. The long description should contain a detailed description of input and output parameters, as well as dependencies such as back-end connectivity. The table below lists three examples:

Copyright © 2001-2011 Voxeo Germany GmbH

Name	Description
Get phone number	Subdialog to obtain the caller's phone number. Includes optional confirmation/correction as well as configurable check of phone number format for country-specific landline/mobile numbers.
Phone banking	Root module for phone banking portal. Provides the welcome section (including account balance), and then branches into sub-applications for transfers and recent transactions.
Jokebox	Module to read out jokes in a variety of categories. Part of an entertainment portal.

Copyright © 2001-2011 Voxeo Germany GmbH

Navigation

Overview

Standard and custom navigation commands enable the caller to control the dialog flow.

Standard navigation commands make the Web functionality of moving back and forward in the dialog available for all channels, i.e. not only in text-based applications but also in voice and video applications.

Custom navigation commands, also to be used in all channels, make the Web functionality of hyperlinks available in voice and video applications. By using an application-specific voice or DTMF command, the caller can, for example, jump back to the main menu, pause the application, or be transferred to a call center agent. These hyperlinks can be accessed throughout the application (through inheritance) or within defined sections of it. Custom navigation commands may or may not undergo a confirmation step, i.e. the Hyperlink object can be configured such that the caller either needs to confirm the activation of the hyperlink (e.g. *Do you really want to leave the application?*), or the target of the hyperlink is processed without confirmation.

In the text and Web channel, custom and standard navigation commands are always visible for the caller, so that they can activate them; in the text channel by responding with the access key defined for the option that represents the command and in the Web channel by activating a Web hyperlink.

The paragraphs Standard Navigation and Custom Navigation below describe how to define navigation using VoiceObjects Desktop. For information on how to define it using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

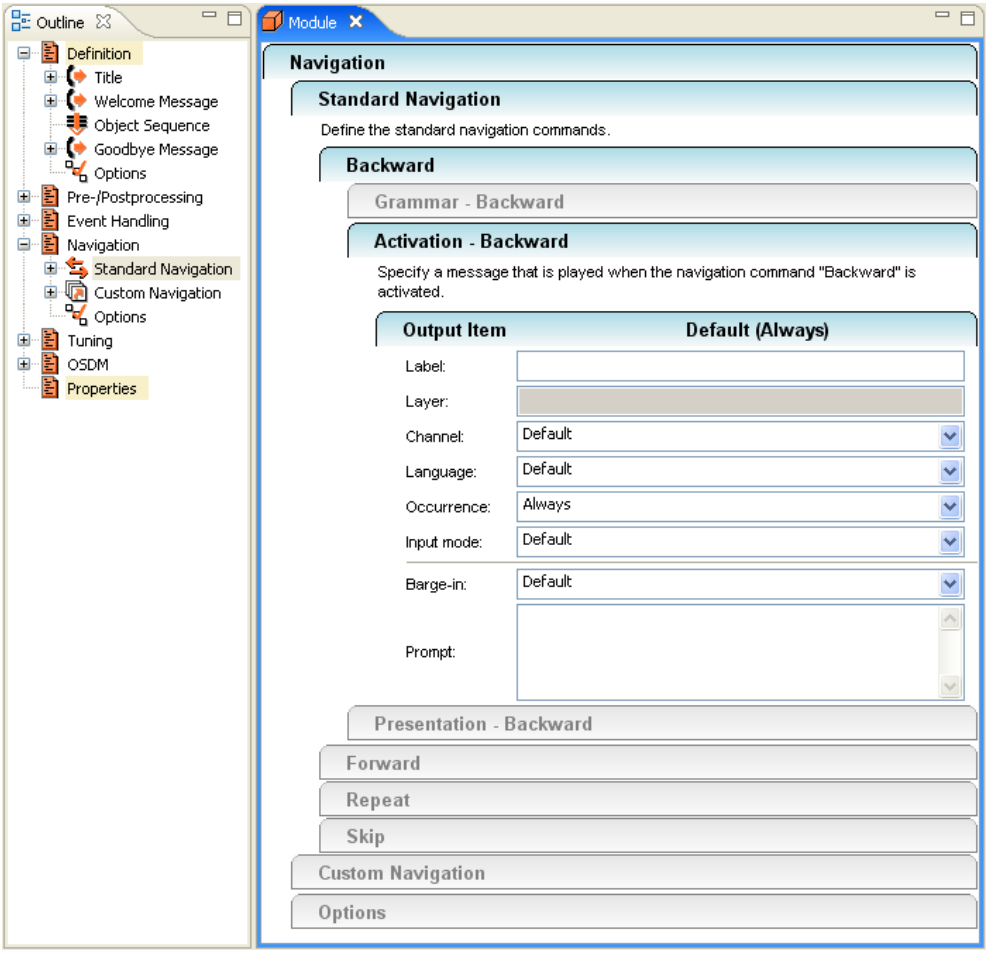
Standard Navigation

Standard navigation provides basic dialog flow navigation, like going back or forward in the dialog. The different types of navigation commands can only be defined in Module objects.

Standard navigation commands specified in a Module object are inherited by the objects in the sequence of this Module object. If Module objects are referenced within other Module objects, the most recent definition takes precedence.

Standard navigation is implemented in the **Standard Navigation** section within the **Navigation** section of the **Module** editor.

Copyright © 2001-2011 Voxeo Germany GmbH



The following standard navigation commands are available, each specified in a separate section (represented by tabs in Desktop for Web):

Navigation	Description
Backward	Moves back one step (i.e. one input state) in time. Any variable assignments, object occurrence counts, etc. are undone.
Forward	Undoes a backward navigation (only available after using backward navigation).
Repeat	Repeats the last output, i.e. the exact same prompt variation played.
Skip	Skips the current object and processes the next object in the dialog flow. Variable objects defined in the result handling of an Input object are not modified.

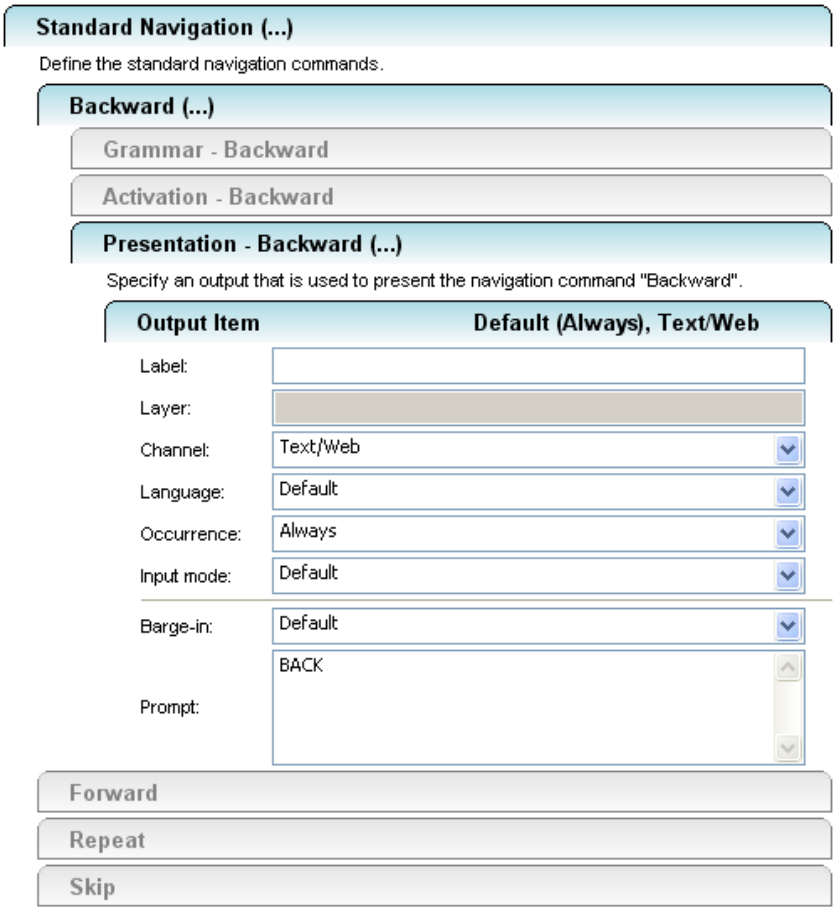
In each section **Backward**, **Forward**, **Repeat**, and **Skip**, three settings can be made.

- The **Grammar** defines the input a caller can make to activate this navigation command. The grammar definition is mandatory in all channels. In the voice and video channel, define voice or DTMF phrases that the caller can say or press. In the text channel, define the access key of the option that the USSD browser generates for the command. In the Web channel, the grammar is conceptually not

Copyright © 2001-2011 Voxeo Germany GmbH

required. But since the definition is mandatory, use question mark “?” as a placeholder definition.

- The **Activation** output can be used to present an output after the caller has activated this command. It is optional in all channels.
- The **Presentation** output defines how to present this command to the caller. In the voice and video channel, this is optional. In the text and Web channel, it is mandatory, as this will be the output that is displayed on the mobile device, so that the caller can activate the command. If there is more than one command defined and you want to specify how the commands are to be ordered, also with respect to custom navigation commands, use the **Label** field of the presentation output to define the criteria for alphanumerical sorting that the server will apply to all active commands in an input state.



For more information on how to define a grammar, refer to the Grammar object in this *Object Reference*.

Information on how to define the outputs can be found in the Output object description.

Note: In the definition of a Service object, you can disable standard navigation altogether. In this case, any commands defined in the **Standard Navigation** section of the respective Module object are not available throughout the application. If standard navigation is enabled in the Service object, a maximum number of navigation steps for standard navigation *backward* must be defined. If the caller activates the *backward* command more often than this maximum number of times, the current object is processed again instead of further going back in the dialog history, and corresponding

Copyright © 2001-2011 Voxeo Germany GmbH

prompts are replayed. For more information on the setting in the Service object, refer to Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*.

To disable a standard navigation command within an application you need to link an autonomous Grammar object over its **Grammar** section. This object must have a precondition that evaluates to false at call time; this will make the whole command inactive. Note that it is not sufficient to just use the layer setting on the embedded grammar to do the same.

The following tables provide some sample grammars for the four available standard navigation commands:

Backward

Grammar	Example

SRGS XML	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<grammar xml:lang="en-US" version="1.0"`
	`root="Backward" mode="voice">`
	`<rule id="Backward" scope="public">`
	`<item>`
	`<count number="optional"> go </count>`
	`</item>`
	`<one-of>`
	`<item> back </item>`
	`<item> backward </item>`
	`</one-of>`
	`</rule>`
	`</grammar>`

Nuance GSL	`;GSL2.0`
	
	`Backward`
	`[`
	`(?go [back backward])`
	`]`

JSGF	`#JSGF V1.0 ISO-8859-1;`
	
	`public <Backward> =`
	`(`
	`[go] back |`
	`backward`
	`);`

Copyright © 2001-2011 Voxeo Germany GmbH

Forward

Grammar	Example
SRGS XML	<?xml version="1.0" encoding="ISO-8859-1"?> <grammar xml:lang="en-US" version="1.0" root="Forward" mode="voice"> <rule id="Forward" scope="public"> <item> <count number="optional"> go </count> </item> <item> forward </item> </rule> </grammar>
Nuance GSL	;GSL2.0 Forward [(?go forward)]
JSGF	#JSGF V1.0 ISO-8859-1; public <Forward> = ([go] forward);

Copyright © 2001-2011 Voxeo Germany GmbH

Repeat

Grammar	Example
SRGS XML	<?xml version="1.0" encoding="ISO-8859-1"?> <grammar xml:lang="en-US" version="1.0" root="Repeat" mode="voice"> <rule id="Repeat" scope="public"> <one-of> <item> <item> repeat </item> <item> <count number="optional"> <one-of> <item> this </item> <item> that </item> </one-of> </count> </item> </item> <item> <one-of> <item> say </item> <item> play </item> </one-of> <item> <count number="optional"> it </count></item> <item> <count number="optional"> once </count></item> <item> again </item> </item> </one-of> </rule> </grammar>

Copyright © 2001-2011 Voxeo Germany GmbH

Grammar	Example
Nuance GSL	;GSL2.0 Repeat [(repeat ?[this that]) ([say play] ?[it this that] ?once again)]
JSGF	#JSGF V1.0 ISO-8859-1; public <Repeat> = ((repeat [this | that]) | ([say | play] [it | this | that] [once] again));

Skip

Grammar	Example
SRGS XML	<?xml version="1.0" encoding="ISO-8859-1"?> <grammar xml:lang="en-US" version="1.0" root="Skip" mode="voice"> <rule id="Skip" scope="public"> <one-of> <item> continue </item> <item> <item> skip </item> <count number="optional"> <one-of> <item> it </item> <item> this </item> <item> that </item> </one-of> </count> </item> <item> go ahead </item> </one-of> </rule> </grammar>

Copyright © 2001-2011 Voxeo Germany GmbH

Grammar	Example
Nuance GSL	;GSL2.0 Skip [continue (skip ?[it this that]) (go ahead)]
JSGF	#JSGF V1.0 ISO-8859-1; public <Skip> = (continue | (skip [it | this | that]) | (go ahead));

Custom Navigation

Custom navigation provides hyperlinks to objects that are accessible throughout the application (or defined sections of it) by specific voice or DTMF commands in the voice and video channel, or by visible options or hyperlinks in the text and Web channel. Each hyperlink is defined by a grammar that activates the hyperlink and the object to be processed. In the text and Web channel, a presentation output is required in addition, so that the caller can activate the command. Instead of processing an object, a hyperlink can also activate an event. If a confirmation of the hyperlink is required, a confirmation request and corresponding grammars must be defined as well.

The designer can specify if the dialog flow returns to the current object after the hyperlink has been processed, if the dialog flow proceeds with the next object, or if the link is without return.

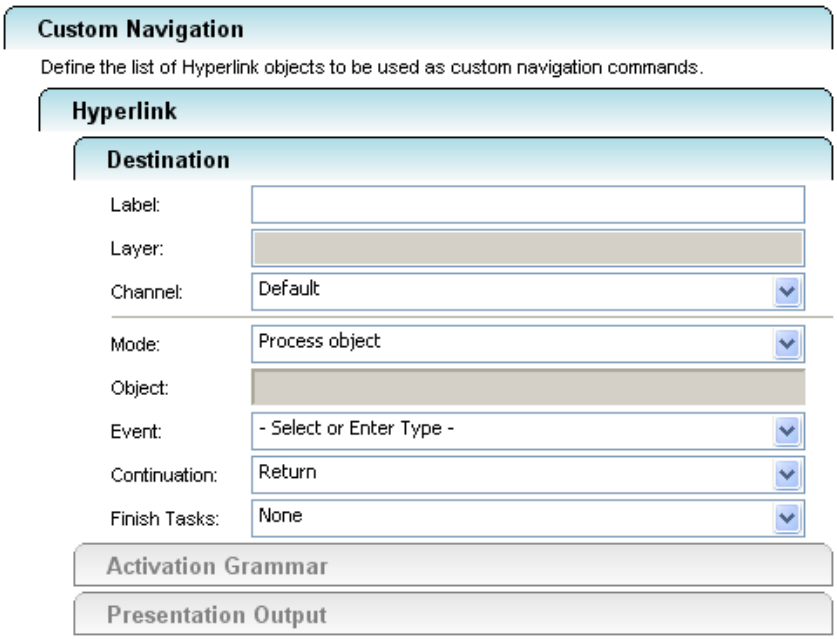
Custom navigation can be defined in Module objects as well as in other objects expecting input from the caller. Hyperlinks defined in any Module object are inherited by the objects in the sequence of that Module object.

Inheritance can be disabled in all objects that offer custom navigation. When switched off, no hyperlinks are inherited by the current object but only locally defined ones are used (if any).

Typical examples for using custom navigation are hyperlinks to the main menu of an application, pausing or exiting an application, transferring to an agent, etc.

Custom navigation commands are implemented in the **Custom Navigation** section within the **Navigation** section of the objects Module, Confirmation, Hyperlink, Input, List, Menu, OSDM, and Plug-In.

Copyright © 2001-2011 Voxeo Germany GmbH



Each custom navigation command consists of an embedded hyperlink definition. This embedded version of a hyperlink is structurally simpler than the autonomous Hyperlink object and only contains an equivalent of the **Destination**, the **Activation Grammar**, and the **Presentation Output** section.

In each **Hyperlink** the following properties can be specified:

Property	Description
Label	Optional parameter to identify the hyperlink in a list.
Layer	Optional parameter to define a layer condition for the specific hyperlink by linking an Expression, Script, Variable, or Collection object, or by selecting a layer condition from the context menu. For more information on layers, see the Layer object in this *Object Reference*, or Chapter 7 – *How to Use Layers* in the *Design Guide*.
Channel	Specifies the channel in which the hyperlink is to be active. Set this to a channel or combination of channels if you only want to activate the command in a specific channel. Otherwise leave it at *Default*.
Mode	Specifies the mode of the hyperlink. Either select *Activate event* and specify an event to be activated in the **Event** field (*Caller - Cancel*, *Caller - Exit*, or *Caller - Help*). Or select *Process object* and specify an object to be processed in the **Object** field. As an example for the first case, the command *help* activates the Help event, which would then be handled by a corresponding event handler in individual Input objects or other objects requesting input from the caller. **Note:** In the text and Web channel, only server-internal events are supported.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Event	Specifies the event to be activated if the mode *Activate event* is selected in the **Mode** field.
Object	Specifies the object to be processed if the mode *Process object* is selected in the **Mode** field.
Continuation	Specifies how to proceed with the dialog flow after the hyperlink has been processed. Select *Return* if you want the dialog to return to the object in which the hyperlink was activated, after processing it. If you do not wish to return to the current dialog flow, select *Do not return*. If you want the current object to be skipped after returning from the hyperlink's subdialog, select *Proceed*. If you merely want the current object to be skipped upon activation of the hyperlink without processing any subdialog, select *Proceed* and leave the **Object** field empty.
Finish Tasks	Defines whether currently active business tasks are to be finished or not when the hyperlink is processed. It can be set to *None* (the default), which means that no task will be finished, *Active*, which means that only the currently active task will be finished, or *All*, which means that all tasks, active or inactive, will be finished. In case one or more tasks get finished, they will automatically be marked with status *Caller Abort*, which means they will be logged as *incomplete* to Infostore. **Note:** This setting is only applicable if **Mode** is set to *Process object*. If **Mode** is set to *Activate event*, the **Finish Tasks** setting has no effect.
Activation Grammar	Specifies the grammar for the input that activates the processing of the hyperlink. For detailed information on defining a grammar, see the Grammar object in this *Object Reference*.
Presentation Output	Specifies how to present the hyperlink to the caller. In the voice and video channel, this output is optional. In the text and Web channel, it is mandatory as it will be displayed on the mobile device so that the caller can activate the command. For detailed information on defining an output, see the Output object in this *Object Reference*.

If you want to define a hyperlink to be confirmed, you have to link an autonomous Hyperlink object into the custom navigation definition, since the embedded version of the hyperlink is restricted and does not provide the confirmation feature.

For detailed information on how to define a Hyperlink object and in particular the confirmation feature, refer to the Hyperlink object.

Note: Custom navigation is intended to allow fast access to major points of the application. Therefore the designer should be careful with the number of hyperlinks and their grammars.
Using a high number of hyperlinks may place a high workload on the media platform in case of voice and video applications. Also try to avoid creating hyperlinks with grammars that are similar to other hyperlinks or other dialog inputs, as this can lead to misrecognitions by the speech recognition engine.

Copyright © 2001-2011 Voxeo Germany GmbH

In the text and Web channel, each navigation command needs to be displayed on the mobile device. As the screen space is most likely rather restricted, do not define too many commands. Typically, a *backward* and *main menu* command is required.

VoiceObjectsXML Definition

Standard navigation

Standard navigation is defined by using the `<standardNavigation>` element, which contains one to four `<standardNavigationItem>` children. Each `<standardNavigationItem>` has a required attribute `direction` taking one of the values *backward*, *forward*, *repeat*, or *skip*. In addition, it has three children:

- <**grammar**> [required]
  Defines the activation grammar for the navigation direction.
- **<output usage="activation">**
  Defines the output to be played before activating the respective standard navigation.
- **<output usage="presentation">**
  Defines the presentation for the standard navigation direction.

Custom navigation

Custom navigation is defined using the `<customNavigation>` element. It has one attribute `inheritance`, which defines whether to inherit hyperlinks from parents objects (*true*, the default) or not (*false*). In addition, it contains one or more `<hyperlink>` element children.

The `<hyperlink>` element has thirteen attributes and seventeen groups of children. Note that the `label`, `layer` and `channel` attributes may only be used when the hyperlink is embedded in `<customNavigation>`, while the children required for confirmation as well as the attributes `recordUtterances` and `recordingScope` may only be used if the hyperlink is defined as an autonomous object.

Attributes

- **mode**
  Defines whether to process an object or to activate an event. Legal values are *object* and *event*. If not specified, defaults to *object*.
- **object**
  Defines the object to be processed if the hyperlink is activated.
- **event**
  Defines the event to be activated if the hyperlink is triggered. Can be any internal VoiceXML event (like *help*, *cancel*, *noMatch* etc.), or an arbitrary custom event name.
- **continuation**
  Defines the continuation mode. Legal values are *return*, *noReturn*, *proceed*. If not specified, defaults to *return*.
- **label**
  A text string providing a name for the hyperlink.
  This attribute may only be used if the hyperlink is directly embedded inside `<customNavigation>`.

Copyright © 2001-2011 Voxeo Germany GmbH

- **`layer`**
  Defines the layer for the hyperlink. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
  This attribute may only be used if the hyperlink is directly embedded inside `<customNavigation>`.

- **`channel`**
  Defines the channel(s) for which this hyperlink is valid. Can be *default*, *voice*, *video*, *text*, *web*, *voiceVideo*, or *textWeb*. If not specified, defaults to *default*.

- **`autoAdvanceNoInput`**
  Define whether the dialog should automatically advance in case of a No Input event. Either *true* or *false*. If not specified, defaults to *false*.
  This attribute may only be used if the hyperlink is autonomous.

- **`autoAdvanceNoMatch`**
  Define whether the dialog should automatically advance in case of a No Match event. Either *true* or *false*. If not specified, defaults to *false*.
  This attribute may only be used if the hyperlink is autonomous.

- **`autoAdvanceDirection`**
  Defines the direction of an auto advance. Can be either *destination* or *origin*. If not specified, defaults to *destination*.
  This attribute may only be used if the hyperlink is autonomous.

- **`recordUtterances`**
  Defines whether to use utterance recording. One of *true*, *false*, *always*, *default*. If not specified, defaults to *false*.
  This attribute may only be used if the hyperlink is autonomous.

- **`recordingScope`**
  Defines the scope for utterance recording. One of *all*, *noMatch*, *recognition*, *default*. If not specified, defaults to *default*.
  This attribute may only be used if the hyperlink is autonomous.

- **`finishTasks`**
  Defines whether business tasks are to be finished or not. One of *none*, *active*, *all*. If not specified, defaults to *none*.

Children

- **`<expression usage=”precondition”>`** or
  `<variable usage=”precondition”>` or
  `<collection usage=”precondition”>` or
  `<script usage=”precondition”>`
  Defines the precondition for the Hyperlink object.

- **`<sequence usage=”preprocessing”>`**
  Defines the preprocessing sequence for the Hyperlink object.

- **`<grammar usage=”activation”>`**
  Defines the activation grammar for the Hyperlink object.

- **`<output usage=”confirmation” type=”initial”>`**
  Defines the “initial” section of the confirmation request.

- **`<output usage=”confirmation” type=”reprompt”>`**
  Defines the “event reprompt” section of the confirmation request.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<output usage="presentation">**
  Defines the presentation for the Hyperlink object.
- **<grammar usage="confirm">**
  Defines the grammar used to confirm the processing of the Hyperlink object.
- **<grammar usage="deny">**
  Defines the grammar used to deny the processing of the Hyperlink Object.
- **<output usage="activation">**
  Defines the activation message.
- **<output usage="deactivation">**
  Defines the deactivation message.
- **<output usage="return">**
  Defines the return message.
- **<output usage="confirmPresentation">**
  Defines the confirm presentation.
- **<output usage="denyPresentation">**
  Defines the deny presentation.
- **<eventHandling>**
  Defines the event handling for the Hyperlink object.
- **<customNavigation>**
  Defines the custom navigation settings for the Hyperlink object.
- **<tuning>**
  Defines the tuning settings for the Hyperlink object.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the Hyperlink object.

Copyright © 2001-2011 Voxeo Germany GmbH

OSDM

Overview

OpenSpeech™ DialogModules™ (OSDMs) are building blocks for voice applications, which accelerate the development process by packaging application functionality for applications based on VoiceXML. They provide all prompts, grammars, dialog flow, and other parameters required to address frequent speech application tasks, like obtaining alphanumeric strings, dates, phone numbers, etc.

The following table lists all thirteen OSDM types available within the VoiceObjects platform together with short descriptions and alias names by which the types are referred to in the OSDM handbook.

Icon	Type	Alias	Description
	*Alphanumeric*	*Alphanum*	Collects a string of connected numbers and letters (i.e. the caller is not required to pause after each digit or letter). Since distinguishing individual letters is especially difficult, it is usually necessary to constrain the task by specifying a list of valid alphanumeric strings (e.g. 1000 possible account IDs).
	*Credit Card Expiration*	*CCExpDate*	Collects the expiration date for a credit card, usually a month and a year. The OSDM recognizes the last day of a given month as well.
	*Credit Card Number*	*CreditCard*	Collects a credit card number. Can understand card numbers for Visa, MasterCard, American Express, Discover, Diners Club, and many private label cards provided by retailers.
	*Currency*	*Currency*	Collects currency amounts in dollars and cents. You can constrain the range of currency amounts, specify the granularity, and specify a disambiguation mode.
	*Custom Context*	*CustomContext*	Allows you to create your own vocabulary and grammar to implement new functions not provided by the other OSDMs.
	*Date*	*Date*	Collects dates in several formats (e.g. “*June fourth*” or “*6/4/70*”). You can specify an entry format and constrain the range of dates accepted by the DialogModule.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Type	Alias	Description
	*Digits*	*Digits*	Fundamentally identical to the Alphanum DialogModule, except that it collects a string of connected digits only. To constrain the recognition task, you can specify a minimum and maximum number of digits to listen for (e.g. at least seven digits, but not more than ten). In most cases, this DialogModule is more accurate than Alphanum for digits-only collections.
	*Number*	*NaturalNumbers*	Collects any numerical amount. Callers can say "*fifty four hundred*" or "*fifty point twenty five*" instead of digits only.
	*Phone Number*	*Phone*	Collects a phone number in the North American Numbering Plan consisting of ten digits (long distance) with optional "1," seven digits (local) or three digits (411, 911, etc.) as configured. Phone numbers must be spoken digit-by-digit; one-eighthundred is supported as a special case. Certain illegal strings for area codes and exchanges are not allowed (e.g., 000-000-xxxx).
	*Postal Code*	*ZipCode*	Recognizes a five-digit or nine-digit US ZIP Code.
	*Social Security Number*	*SocialSecurity*	Collects a 9-digit US social security number. Illegal numbers, such as those beginning with three zeroes, are screened out.
	*Time*	*Time*	Collects a time of day. Callers can use 12-hour or 24-hour times, as well as prefix words such as "*about*" and "*around*" To constrain the recognition task, you can specify a range of valid times, granularity, and a disambiguation mode for determining whether to listen for times in the past or in the future.
	*Yes/No*	*YesNo*	Collects either an affirmative or negative response from the caller.

An OSDM dialog flow is comprised of a collection phase and a confirmation phase. During the collection phase a specific item type (e.g. a date or a string of digits) is collected from the caller. The subsequent confirmation phase is entered only if recognition confidence during the collection phase is in the midrange between the confidence level and the high confidence threshold, or if the confirmation mode is explicitly set to confirm ALWAYS.

For more information on OSDMs in general, refer to the OSDM handbook and the OSDM Website.

The OSDM object is only designed for the voice channel; in all other channels it will not be processed. For further information on channels refer to Chapter 10 – *How to Support Multiple Phone Channels* in the *Design Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

The Object Definition below covers the configuration of the OSDM object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The thirteen different OSDM object types belong to the object category OSDMs.

Dialog Flow Scenario

The following dialog flow example is based on the OSDM type *Date*, which asks the caller for the date of departure and return in a flight information service scenario.

Object – Caller	Dialog Flow
	*Please tell me the date of your departure.*
 Caller	*The twenty-second of November.*
	*(confidence is high enough, proceeding to next OSDM) And when do you want to return?*
 Caller	*That's the first of December, please.*
	*(confidence is above the confidence level but below the high confidence threshold) Not sure I got you correctly. Was that the first of December?*
 Caller	*Yes.*
	*(checking available flights)*
	*I'm sorry, there are no flights available any more for the given dates.*

In this example, the first utterance receives a confidence value that is high enough to simply proceed in the dialog, skipping the confirmation phase. In the second OSDM, the utterance needs to be confirmed.

The second example describes a simple dialog requesting the name of a person to call in an auto-attendant application scenario. The OSDM type used is *Custom Context*.

Object – Caller	Dialog Flow
	*Thank you for calling. Please say the name of the person or department you'd like to reach.*
 Caller	*(doesn't say anything)*
	*Sorry, I didn't hear you. Please tell me with whom you would like to speak.*
 Caller	*Well, I'd sure like to talk to Mark if he's around. Is he?*

Copyright © 2001-2011 Voxeo Germany GmbH

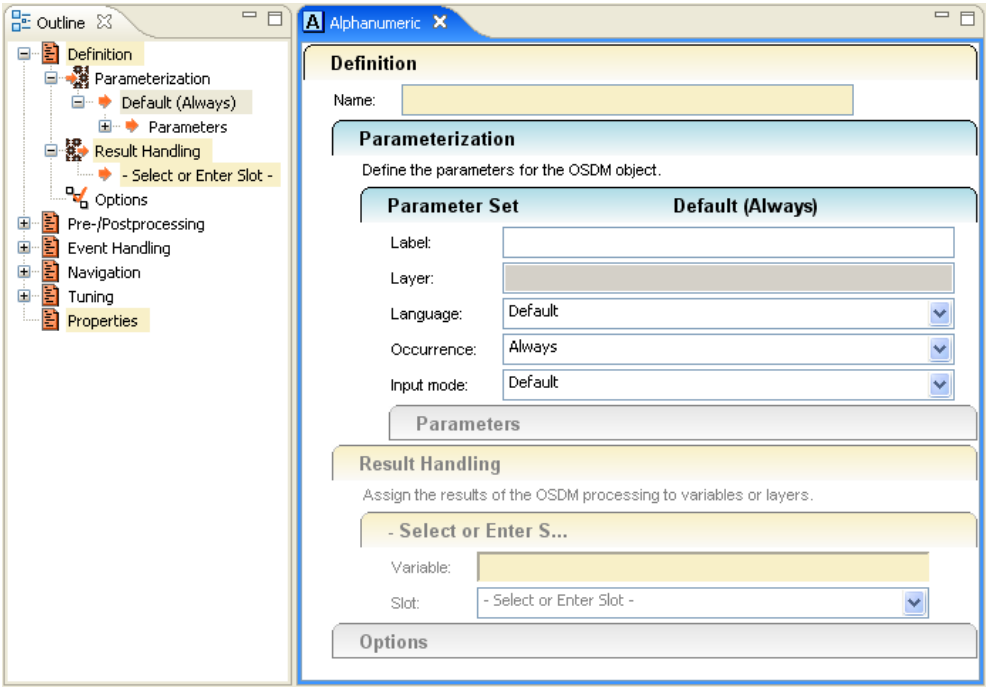
Object – Caller	Dialog Flow
	Sorry, I didn't understand. Please say just the first and last name of the person you want to speak with.
Caller	*Mark Levinson.*
	(not entirely sure) Was that "Mark Levinson?"
Caller	*Yes.*
	Thanks for being patient. Please hold while I check to see if he's available.

Note that all interaction is taking place within one and the same OSDM object before proceeding with the dialog flow holding more OSDMs or other objects.

Object Definition

All thirteen OSDM types share the same editor layout. The **Definition** of each OSDM object type provides the following sections:

- Parameterization
  To specify the list of parameters to be passed to the OSDM.
- Result Handling
  To specify where to store the result coming from the OSDM processing.
- Options
  To configure processing options.



Copyright © 2001-2011 Voxeo Germany GmbH

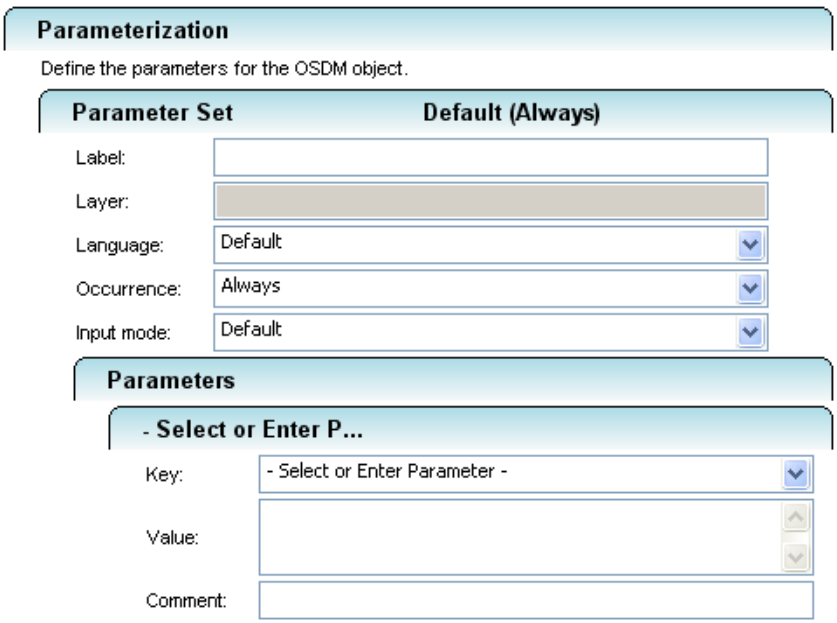
The **OSDM** editor provides individual event handling, navigation, and tuning which is described in Additional Object Configuration below.

For further details regarding the additional object configuration, refer to *Pre-/Postprocessing*, *Event Handling*, *Navigation*, *Tuning*, and *Properties* in this *Object Reference*.

Parameterization

Within the **Parameterization** section, one or more **Parameter Set** items can be defined.

An OSDM object may contain any number of individual parameter set items. Each parameter set item can hold one or more parameters.



You can use the items to group parameters of the same kind (e.g. prompts, logging settings), or to group parameters to be active for different languages, input modes, custom layers etc.

Note: In case more than one parameter set item is active, i.e. remains after filtering of language, input mode, custom layer, and occurrence level, all items are rendered, instead of picking one item randomly. This is different from the behavior found in Output objects, e.g., where random prompting makes sure to pick only one item at a time, with exhaustive randomization when revisiting the object.

OSDM parameters can also be set in Module objects. This allows you to define general properties that are to be valid across all OSDM objects (such as default directories) at the module level, and use inheritance to propagate these parameters down to the OSDM object level.

In the **Parameter Set** section, the following properties can be specified for each parameter set item:

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Label	Optional parameter to identify the parameter set in a list.
Layer	Optional parameter to indicate the custom layer that the parameter set item belongs to. For more information on layers, see the *Layer* object in this *Object Reference* or Chapter 7 – *How to Use Layers* in the *Design Guide*.
Language	To identify the language in which the respective parameter set item is provided. If an item is appropriate in all relevant languages, the default language layer setting *Default* should be used. For languages that are used in multiple countries, both the specific language options (e.g. *English (SG), English (US)*) and the generic option (*English*) are provided. **Note:** OSDM objects currently only support the following languages: Cantonese Chinese (Hong Kong), German (Germany), English (Australia), English (United Kingdom), English (Singapore), English (US and Canada), Spanish (Mexico), Spanish (US), French (Canada), French (France), Japanese (Japan), and Mandarin Chinese (Taiwan).
Occurrence	Enables you to use different parameter sets depending on how often the OSDM object has already been visited. *Always* means that regardless of the number of previous visits the respective parameter set item is always active. *Only once* means that the parameter set item is active only the first time the OSDM object is visited. And *If >= N* means that the item is active only if this is at least the Nth time that the OSDM object has been visited.
Input mode	To associate the item with an input mode, so that different parameters can be passed to the OSDM depending on the currently active input mode. This is typically used for applications that are designed to be controlled with both input modes: *Voice* and/or *DTMF*. Using this setting, you can, for example, define one prompt that is played when the input mode is set to *Voice* (*Please say yes or no*), and a different one for input mode *DTMF* (*Please press 1 for yes and 0 for no*), both in the same OSDM object. You can set the input mode either to *Voice*, *DTMF*, or *Voice+DTMF*. If you do not want to distinguish your parameter set items with regard to input mode, leave the setting at *Default*. *Default* indicates that the input mode is defined within the corresponding service. For more details, refer to Configuring a Service in Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*.

Additionally, one or more parameters can be defined in the **Parameters** section within the **Parameter Set** section by specifying the following properties:

Copyright © 2001-2011 Voxeo Germany GmbH

Key
Holds the parameter name itself. You can either select an entry from the available drop-down list, or enter a custom value. Instead of entering custom values from scratch, you would typically want to select an entry first and then modify it, i.e. change the index of one of the entries providing a default index of 1 (like collection_grammar1). For further information on how to do this, refer to the description of list fields in Chapter 6 – *Object Editors* either in the *Desktop for Eclipse Guide* or in the *Desktop for Web Guide*. For details on the meaning of each individual parameter, refer to the OSDM handbook.

Note: The set of entries in this list depends on the type of OSDM selected, i.e. you won't find all of the OSDM parameters in each **OSDM** editor.

Value
Enter the value for the parameter selected in the **Key** field. This can be an arbitrary string or a combination of strings and references to existing objects. The type of object to be linked into the **Value** field depends on the parameter selected. For a prompt-related parameter like *collection_initialprompt*, you can e.g. link an Output or Audio object. For a resource-related parameter like *applicationgrammarsdirectory*, you can link a Resource Locator object, etc.

Existing objects can be inserted by typing in their names using a shortcut notation. For more information on how to do this, refer to Chapter 6 – *Object Editors* in the *Desktop for Eclipse Guide* and *Desktop for Web Guide*, respectively.

The following table lists all possible categories of OSDM parameters, together with example parameters and the types of objects that can (and should only) be used for the corresponding category. Note that technically all linked objects will be rendered to a string value, so, for a given parameter, referencing an object that is not among the following list might lead to invalid parameters.

Parameter Category	Example Parameters	Possible Objects
Data	entries	Variable Expression Script Collection Layer
Directory	applicationpromptsdirectory, baselinegrammarsdirectory	Resource Locator Variable Expression Script Collection Layer
Grammar	dtmfyes, collection_grammar	Grammar Variable Expression Script Collection Layer
Limit	maxinvalidanswers, maxhelpcount	Variable Expression Script Collection Layer

Copyright © 2001-2011 Voxeo Germany GmbH

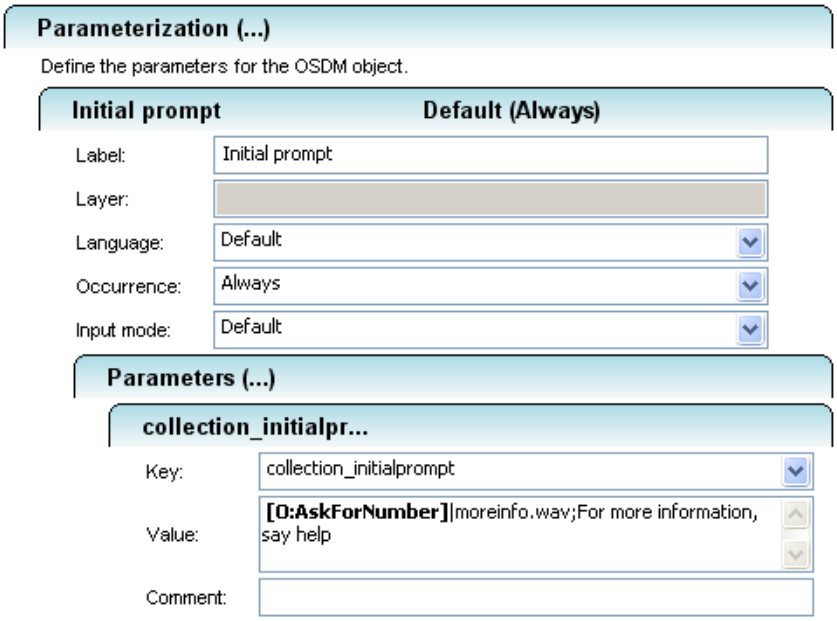
Parameter Category	Example Parameters	Possible Objects
Logging	debuglevel, eventsLogFile	Variable Expression Script Collection Layer
Option	length, dtmfdateformat, maxextension	Variable Expression Script Collection Layer
Prompt	collection_initialprompt, invalidanswerprompts	Output Audio Silence Variable Expression Script Collection Layer
Property	property_collection	Variable Expression Script Collection Layer
Recognition	avoidsamemistake, highconfidencelevel	Variable Expression Script Collection Layer
Rendering	lang, vxmlspec	Variable Expression Script Collection Layer
Setup	recognizer, waveformcapture	Variable Expression Script Collection Layer

Note: If an Output object is linked into the **Value** field that supports random prompting, only one variation is taken and passed into the OSDM object.
If a Grammar object is linked into the **Value** field and more than one Grammar item is active while processing the OSDM object, i.e. more than one Grammar item remains after filtering by language and custom layers, only the first item (according to the order of the definition in the corresponding **Grammar** editor) is passed to the OSDM object. This is because OSDMs currently allow defining only one grammar for grammar-related parameters. Note that if you use Grammar objects for hyperlinks through the **Navigation** definition, there is no such restriction; all Grammar items remaining after filtering are passed to the OSDM.

Copyright © 2001-2011 Voxeo Germany GmbH

The OSDM parameters follow a given formatting convention, which allows to combine audio file references with alternative TTS text, and to provide more than one value for a prompt-related parameter at the same time. A list of audio file URLs can be defined by using the pipe symbol ("|") to separate the individual files, and by using the semicolon (";") to add alternative text to be read out by TTS in case the audio file is missing or corrupt. Note that when linking existing Output or Audio objects into the **Value** field, the server takes care of producing the correct syntax for the OSDM parameters. In case you need to add additional values, make sure to use the symbols required for the corresponding parameter correctly. Furthermore, if you link Variable, Expression, Script, Collection or Layer objects, you also need to make sure that their value has the correct syntax.

The following example shows the definition of the parameter *collection_initialprompt*, which consists of a reference to an existing Output object, followed by another audio file URL with alternative text:



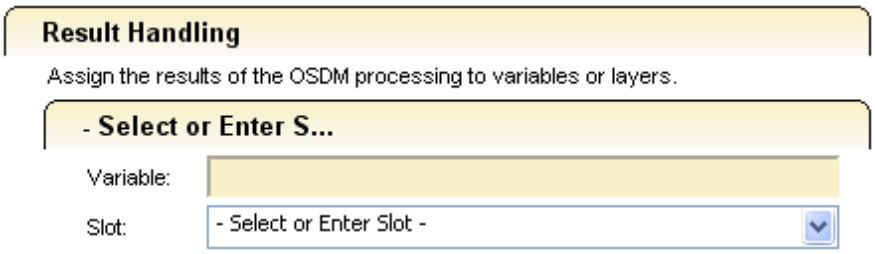
Note: Variable, Expression, Script, Collection, and Layer objects have (embedded or autonomous) Format objects associated with them. This formatting has effect on the way their respective values are rendered for playback. When linking any of these objects into the **Value** field of a **Parameters** section, any formatting is ignored. If you want to make use of formatting, you need to wrap the corresponding object in an Output object and link this Output object instead.

Comment
Enter a comment describing the parameter, if required. This field is optional.

Copyright © 2001-2011 Voxeo Germany GmbH

Result Handling

The **Result Handling** section defines one or more result handling items that assigns return values from the OSDM to Variable or Layer objects.

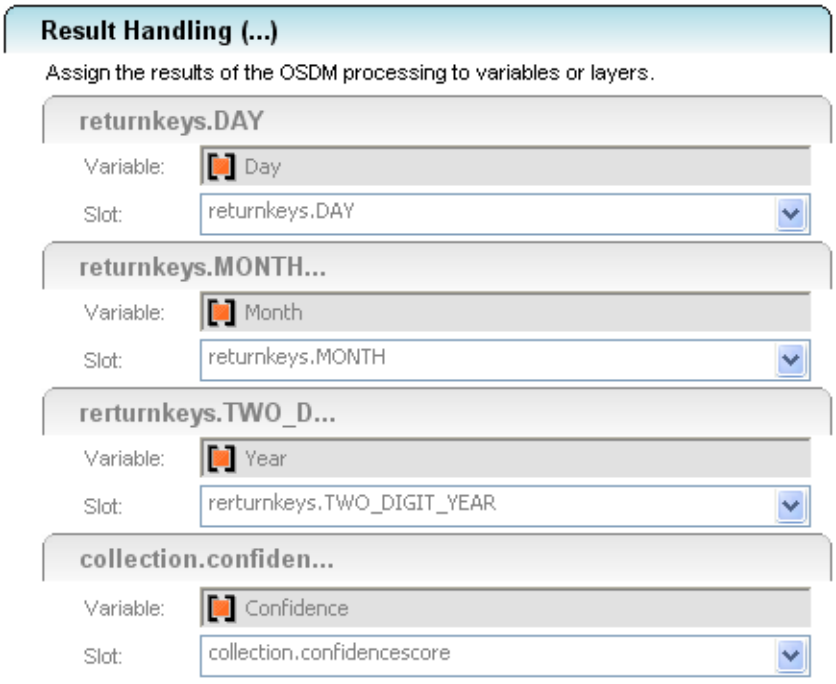


The **Slot** field allows to either type in a custom slot name, or to select one of a set of predefined slots from a drop-down list.

Note: The set of entries in this list depends on the type of OSDM selected, i.e. you won't find all of the slot names in each OSDM editor.

Typical return values are the result itself (the semantic interpretation), statistical information on the number of No Input or No Match events that occurred within the OSDM, the raw utterance of the caller, confidence values etc. For a full list of possible slot names, refer to the OSDM handbook.

The following example shows the definition of the **Result Handling** of an OSDM object of type *Date*, which returns day, month, and year, as well as the confidence score of the collection phase.



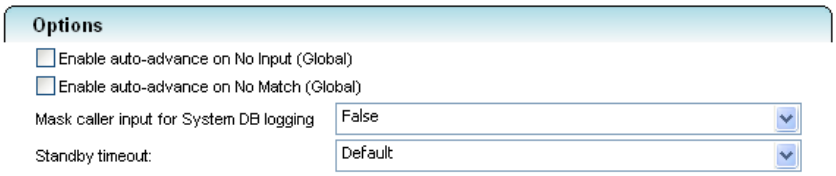
You can also retrieve ASR-related information regarding confidence values, utterances and more using the system function LASTRESULT within an Expression object.

Copyright © 2001-2011 Voxeo Germany GmbH

Event-related information can be retrieved using EVENTCOUNT and EVENTLIST. For a description of these functions, refer to the Expression object in this *Object Reference*.

Options

The **Options** section of the OSDM object specifies four processing options:



Enable auto-advance on No Input (Global) enables auto-advance in case no response from the caller is detected in the collection or confirmation phase. If the check box is selected, a No Input event in either the collection or confirmation phase finishes the processing of the OSDM object and proceeds to the next object in the dialog flow instead of processing the event handling specifications. By default, the check box is clear.

Enable auto-advance on No Match (Global) enables auto-advance in case the caller says something that cannot be matched with the grammar in the collection or confirmation phase. If the check box is selected, a No Match event in either the collection or confirmation phase finishes the processing of the OSDM object and proceeds to the next object in the dialog flow instead of processing the event handling specifications. By default, the check box is clear.

Note: The options are for your convenience only. The same effect can be achieved by specifying a No Input or No Match event handling for level 1 and continuation set to *Proceed*. Use the options described above if you want to proceed on the first No Input or No Match event. If you want to proceed at a later event occurrence level, specify the appropriate event handlers. Using a dedicated event handler also allows you to process any object before proceeding, e.g. play a prompt through an Output object. Also note that the options described above affect both the collection and the confirmation phase. If you want different auto-advance behavior for the two phases, don't use these options but define corresponding event handlers for the two phases instead and set continuation to *Proceed*.

Mask caller input for System DB logging encrypts the caller input before writing it to the system database via Infostore, the logging component of VoiceObjects Server. This is useful for sensitive data, like PINs, which mustn't be stored as plain text for security reasons.
The default setting *False* indicating that the data is not encrypted. *True* indicates that data is encrypted in such a way that while the original value cannot be recovered, different input values still produce different encrypted values so that subsequent analyses are possible. *Complete* indicates that the values are completely masked by a single static string. (By default the string "***MASKED***" is used; it can be configured in *components_VOServer.xml* using the `<maskUserInputComplete>` element.)

The **Standby Timout** field specifies the maximum time for which the dialog session is kept alive while the OSDM is processed. By default this timeout is set to *Default*, which indicates that the value is taken from the dialog context. For more details, refer to Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*. When using a Variable, Expression, Script, or Layer object to define the standby timeout dynamically at call time, or when typing in a custom value instead of selecting one of

Copyright © 2001-2011 Voxeo Germany GmbH

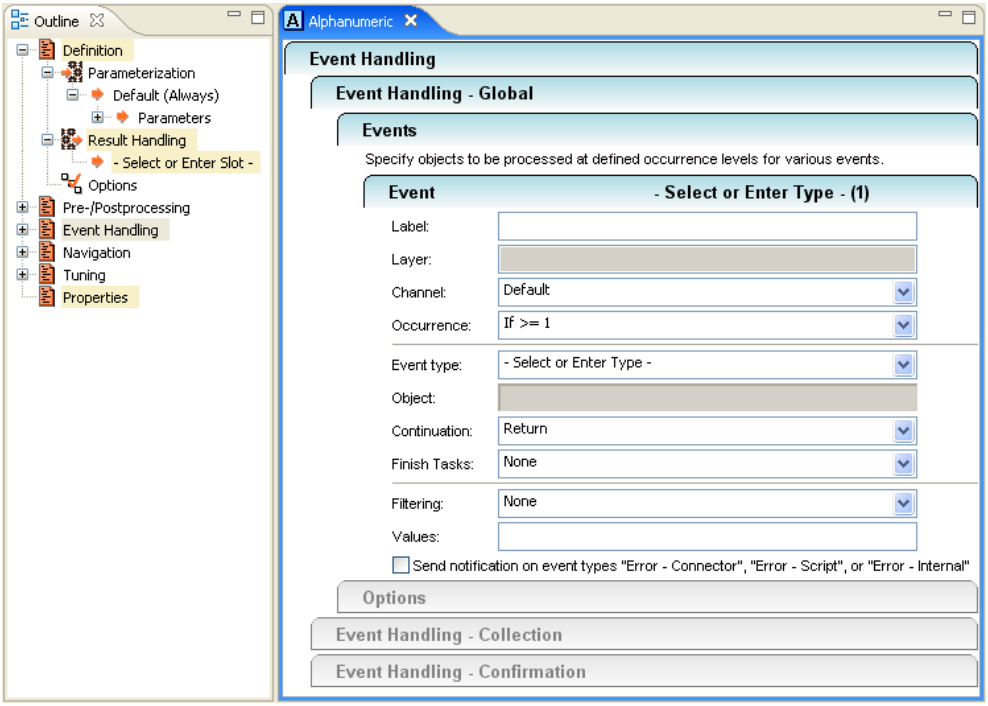
the values from the drop-down list, the corresponding value needs to be a positive number and will be applied as seconds, e.g. for a custom timeout of 5.5 minutes, type in 330.

Note: If a caller spends time in the OSDM longer than the standby timeout, a return to the VoiceObjects session is not possible. Thus the developer needs to make sure that the standby timeout is at least as long as the maximum duration of the OSDM.

Additional Object Configuration

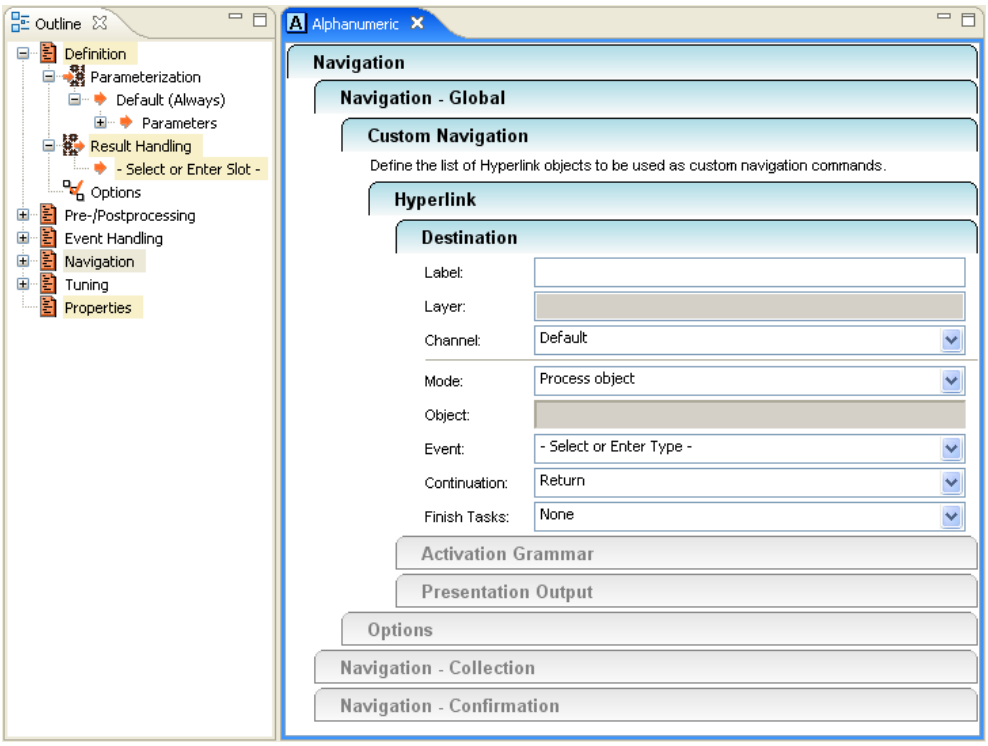
The **Event Handling**, **Navigation**, and **Tuning** sections of the **OSDM** editor each provide a global section which applies both to the collection and the confirmation phase, as well as separate definitions for the collection and the confirmation phase.

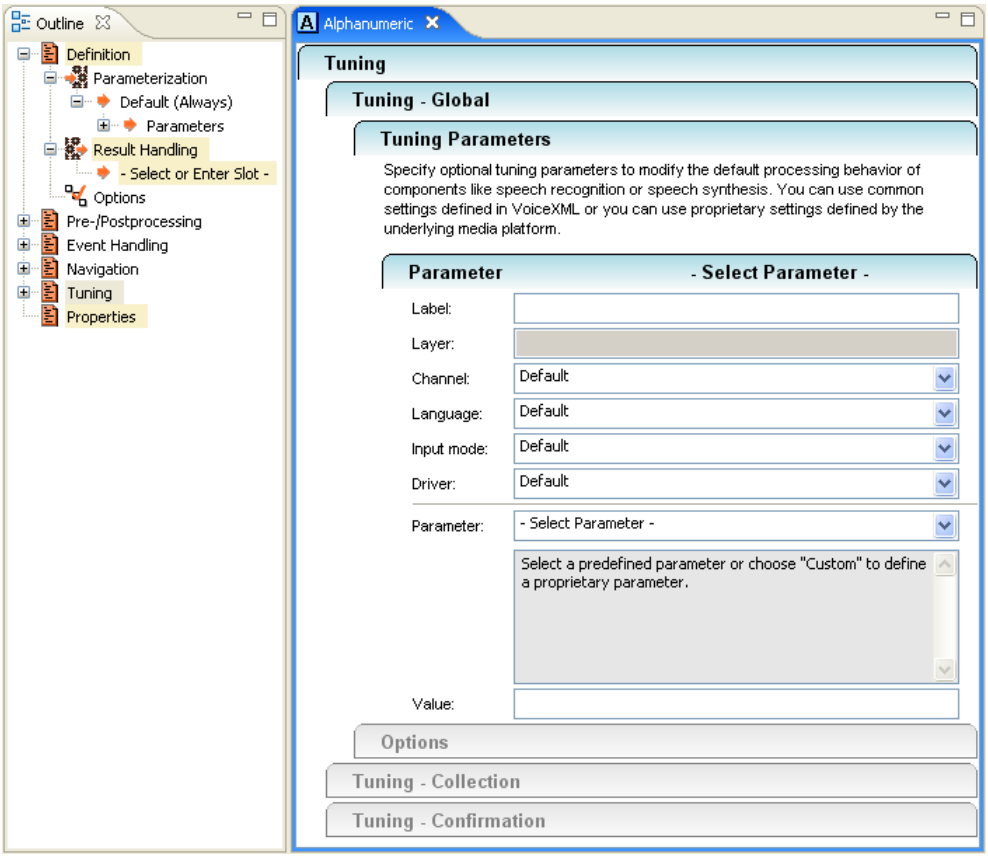
If you do not want to define separate event handling, navigation, or tuning for the different phases of the OSDM, leave the corresponding sections empty and just define the **Global** section.



All event handlers that are inherited from parent objects are taken as if they were defined in the **Global** section, i.e. they are active both for the collection and the confirmation phase. If you want different event handling in one of the two phases, you can define it in the corresponding section and thus overwrite the inherited settings. If you define event handlers or tuning parameters both in the **Global** and in any of the other two sections, the more specific sections take precedence over the settings in the **Global** section. As for navigation, embedded hyperlinks defined in the **Global** section and those defined in the **Collection** or **Confirmation** section will all be active at the same time.

Copyright © 2001-2011 Voxeo Germany GmbH





Copyright © 2001-2011 Voxeo Germany GmbH

For further details regarding the additional object configuration, refer to *Pre-/Postprocessing*, *Event Handling*, *Navigation*, *Tuning*, and *Properties* in this *Object Reference*.

Event handling is usually defined in OSDMs by setting the relevant parameters for event types, prompts to be played, etc. This could be done by providing the corresponding parameters in the **Key** field within the **Parameter** section of a parameter set in the **Definition** section, although these parameters are not part of the available list of keys for each OSDM. When defining event handling, it is strongly recommended to use the **Event Handling** section instead, which leverages all important VoiceObjects concepts such as random prompting or inheritance. Note, that if event handlers are defined both through parameters and in the **Event Handling** section, the parameters take precedence over any settings made in the **Event Handling** section.
The same holds true for navigation and tuning.

Note: Grammars for hyperlinks usually don't require setting a slot, and some platforms even reject hyperlink grammars containing slot assignments. In OSDM objects, however, the grammar associated with a hyperlink always needs to set a slot with the predefined slot name *dm_command*. This is necessary, in order to tell the server which hyperlink to activate when returning from the OSDM processing, since all communication between the OSDMs and VoiceObjects Server takes place through the slots.
If you want to use a Hyperlink object through the **Navigation** section, you can either use TTG, embedded, or external grammars. In case of TTG, you don't have to deal with slot assignments, since the server does this automatically when generating the final grammar at call time. In case of embedded grammars, you have to define a label for this hyperlink in its **Label** field and use this label as the slot value for the slot *dm_command*. Alternatively, you can use the shortcut notation *@HYPERLINK@*, which will be resolved at call time. In case of external grammars, setting the **Label** field on the Hyperlink object is required, and the slot *dm_command* must be set to this label. Note, that when using autonomous Hyperlink objects, the object name must be taken instead of the label.

Since the activation of the hyperlink might need to be confirmed, you have to provide an additional slot *dm_confirm_string* in the hyperlink grammar that defines the utterance that will be used within the confirmation prompt to repeat what the caller has said. The value of this slot is typically the same as the utterances defined in the grammar. Note again that setting this slot is not necessary for TTG grammars.

Note: For technical reasons, the two phases of OSDM objects, collection and confirmation, are hidden from the server, i.e. they are treated as one dialog step. This has effects on standard and custom navigation. For the standard navigation commands *Backward* and *Forward* this means that an OSDM is always entered in the collection phase, never in the confirmation phase. Thus, going backward on the object immediately following an OSDM object in the dialog flow always leads to the collection phase, even if the confirmation phase was the last step. Going backward within the confirmation phase leads to the input state prior to the OSDM object.
The same holds true for custom hyperlinks activated within the confirmation phase, and event handlers that process a Goto object.

Copyright © 2001-2011 Voxeo Germany GmbH

VoiceObjectsXML Definition

The OSDM object types are represented by the VoiceObjectsXML element `<osdm>`. It has four attributes and fourteen groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<osdm>` element uses the embedded `<osdmItems>` element.

OSDM

Attributes

- **`type`**
  Defines the type of the OSDM object. Possible values are *alphanumeric*, *creditCardNumber*, *creditCardExpiration*, *currency*, *customContext*, *date*, *digits*, *number*, *phoneNumber*, *postalCode*, *socialSecurityNumber*, *time*, *yesNo*.
  If not specified, defaults to *customContext*.
- **`autoAdvanceNoInput`**
  Defines whether the dialog should automatically advance in case of a No Input event. Either *true* or *false*. If not specified, defaults to *false*.
- **`autoAdvanceNoMatch`**
  Defines whether the dialog should automatically advance in case of a No Match event. Either *true* or *false*. If not specified, defaults to *false*.
- **`maskCallerInput`**
  Defines whether the input made by the caller should be masked when writing it to Infostore. This is recommended for sensitive data such as e.g. banking account numbers or PINs.
  One of *true*, *false*, or *complete*. If not specified, defaults to *false*.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the OSDM object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the OSDM object.
- **`<osdmItems>`**
  Defines the parameterization for the OSDM object.
- **`<resultHandling>`**
  Defines the result handling for the OSDM object.
- **`<eventHandling usage="global">`**
  Defines the event handling for both phases of the OSDM object.
- **`<eventHandling usage="collection">`**
  Defines the event handling for the collection phase of the OSDM object.
- **`<eventHandling usage="confirmation">`**
  Defines the event handling for the confirmation phase of the OSDM object.
- **`<customNavigation usage="global">`**
  Defines the navigation settings for both phases of the OSDM object.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<customNavigation usage="collection">**
  Defines the navigation settings for the collection phase of the OSDM object.
- **<customNavigation usage="confirmation">**
  Defines the navigation settings for the confirmation phase of the OSDM object.
- **<tuning usage="global">**
  Defines the tuning settings for both phases of the OSDM object.
- **<tuning usage="collection">**
  Defines the tuning settings for the collection phase of the OSDM object.
- **<tuning usage="confirmation">**
  Defines the tuning settings for the confirmation phase of the OSDM object.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the OSDM object.

Example

```
<osdm type="phoneNumber">
    <osdmItems>
        <osdmItem label="Prompts">
            <osdmParameter key="collection_initialprompt">
                <output link="#Ask for phone number"/>
            </osdmParameter>
            <osdmParameter key="confirmation_initialprompt">
                <output link="#Confirm phone number"/>
            </osdmParameter>
        </osdmItem>
        <osdmItem label="Grammars">
            <osdmParameter key="grammar">
                <grammar link="#Phone numbers"/>
            </osdmParameter>
        </osdmItem>
    </osdmItems>
    <resultHandling>
        <item alias="returnvalue" object="#Phone number"/>
    </resultHandling>
</osdm>
```

OSDMItems

Attributes

- **inheritance**
  Defines whether OSDM parameter settings inherited from parents objects are used (*true*, the default) or not (*false*). This is only relevant for an `<osdmItems>` element used within a `<module>` element.

Copyright © 2001-2011 Voxeo Germany GmbH

Children

- **+<osdmItem>**
  Defines the list of OSDM items.

Example

```
<osdmItems>
    <osdmItem label="Prompts">
        <osdmParameter key="collection_initialprompt">
            <output link="#Ask for phone number"/>
        </osdmParameter>
        <osdmParameter key="confirmation_initialprompt">
            <output link="#Confirm phone number"/>
        </osdmParameter>
    </osdmItem>
    <osdmItem label="Grammars">
        <osdmParameter key="grammar">
            <grammar link="#Phone numbers"/>
        </osdmParameter>
    </osdmItem>
</osdmItems>
```

OSDMItem

Attributes

- **label**
  A text string providing a name for the OSDM item.
- **language**
  Defines the language for which this OSDM item is valid. Can be *default* or a valid language code (e.g. *de-DE*, *en-US*, etc.). If not specified, defaults to *default*. Appendix A – Language Codes contains a list of all language codes available in VoiceObjects together with the respective language they represent.
- **occurrence**
  Defines the occurrence level for which this OSDM item is valid. Can be *always*, *once*, or an integer between 1 and 10. If not specified, defaults to *always*.
- **inputMode**
  Defines the input mode for which this OSDM item is valid. Can be *default*, *voice*, *dtmf*, or *voicedtmf*. If not specified, defaults to *default*.
- **layer**
  Defines the layer for the OSDM item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form "Layer=State" or "Layer!=State" where "State" is the label of a state for the layer "Layer".

Copyright © 2001-2011 Voxeo Germany GmbH

Children

- **+<osdmParameter>**
  Defines the list of parameters in this item.

Example

```
<osdmItem label="Prompts">
    <osdmParameter key="collection_initialprompt">
        <output link="#Ask for phone number"/>
    </osdmParameter>
    <osdmParameter key="confirmation_initialprompt">
        <output link="#Confirm phone number"/>
    </osdmParameter>
</osdmItem>
```

OSDMParameter

Attributes

- **key [required]**
  A text string providing the parameter key.
- **comment**
  A text string providing a comment describing the parameter.

Children

- Any number of the following:
  <variable>, <expression>, <script>, <layer>, <text>, <resourceLocator>, <audio>, <output>, <silence>, <grammar>
  Defines the value to be passed to the OSDM for the respective parameter key.

Example

```
<osdmParameter key="collection_initialprompt">
    <output link="#Ask for phone number"/>
</osdmParameter>
```

Object Interoperability

The following table contains all object types that can reference an OSDM object:

Icon	Object Name	Use Case Example
	Module	An OSDM object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, an OSDM object is used within a sequence in a dialog flow.
	Menu	An OSDM object can be used as the destination object within a Menu item.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Object Name	Use Case Example
	Confirmation	An OSDM object is frequently used as the destination object within a correction item inside a Confirmation object.
	If	An OSDM object can be used in either the THEN item or the ELSE item of an If object.
	Case	An OSDM object can be used in any WHEN item and the ELSE item of a Case object.
	Loop	An OSDM object can be used within the embedded sequence of a Loop object.
	Goto	An OSDM object can be referenced via a Goto object.
	Hyperlink	An OSDM object can be the destination of a hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of an OSDM object should indicate which type of input is requested from the caller. The short description should contain a brief explanation of the parameters defined for the OSDM. The table below lists three examples:

Name	Description
 Ask for phone number	Requests a phone number from the caller and places it in the variable *Phone Number*.
 Confirm transaction	Replays the transaction to be performed and asks for an explicit yes/no confirmation. The response is stored in the variable *Yes/No Answer* as *yes* or *no*.
 Ask for car make	Asks for the car manufacturer. Results are stored in variable *Car make*.

Copyright © 2001-2011 Voxeo Germany GmbH

Output

Overview

The Output object is a configurable dialog component to present an output to the caller. In the voice and video channel, outputs may contain text that is read out through a text-to-speech engine, or prerecorded audio or video files, possibly containing various dynamic content as well as Silence objects. In the text and Web channel, the output defines what is displayed on the screen of the mobile device. References to Audio, Video or Silence objects will be ignored in these channels.

The Object Definition below covers the configuration of the Output object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Output object belongs to the object category Components.

Dialog Flow Scenario

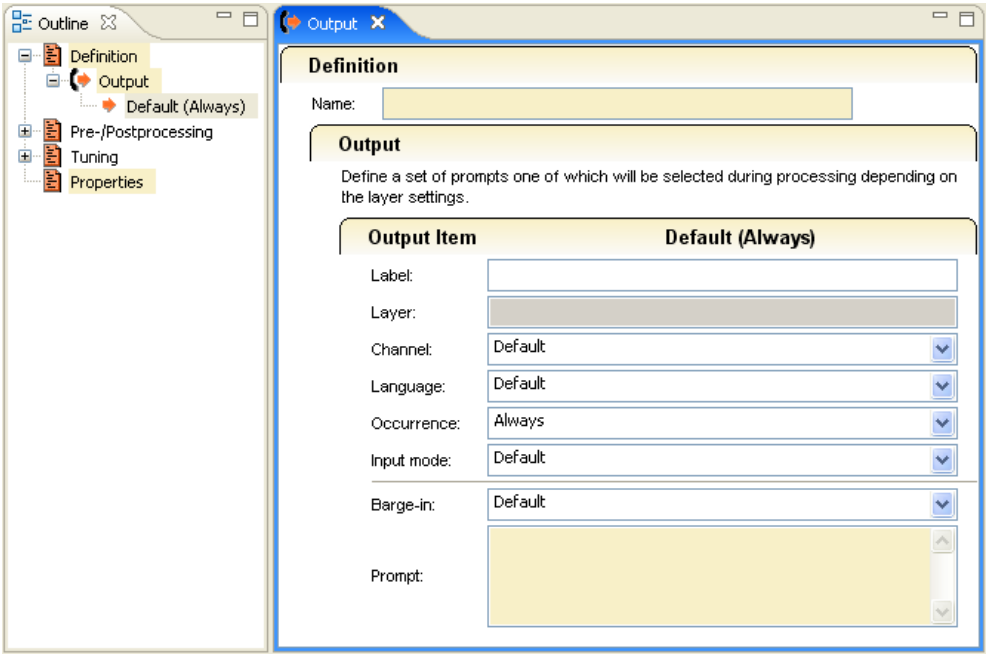
The following dialog flow example describes a simple output from a flight booking application.

Object – Caller	Dialog Flow
	Thanks for booking flight 456 from Boston to Chicago.

Object Definition

The **Definition** of the Output object provides one section:

- Output
  To define the set of Output items.



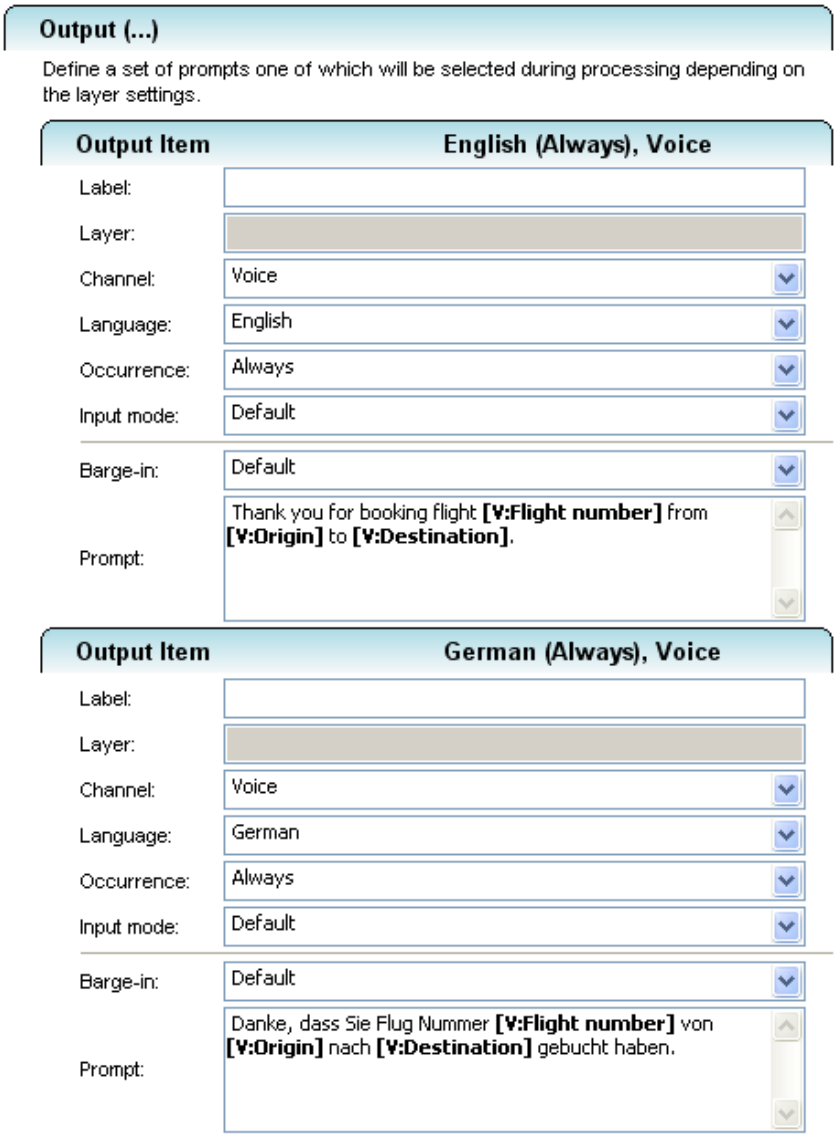
Copyright © 2001-2011 Voxeo Germany GmbH

For information regarding additional object configuration refer to *Pre-/Postprocessing, Tuning* and *Properties* in this *Object Reference*.

Note: If an Output object is used within an event handling definition, any pre- or postprocessing defined for that Output object is ignored. This is also the case when a Goto object with an Output object as its destination object is linked into an event handling definition.
An Output object's pre- and postprocessing is also ignored if the object is referenced within another Output object.

Output

An Output object may contain any number of individual Output items.



Copyright © 2001-2011 Voxeo Germany GmbH

In each **Output Item** the following properties can be specified:

Property	Description
Label	Optional parameter to identify the Output item in a list.
Layer	Optional parameter to indicate the custom layer that the Output item belongs to. For more information on layers, see Chapter 7 – *How to Use Layers* in the *Design Guide*.
Channel	To identify the channel in which the respective Output item is to be active. If an Output item is appropriate for all channels, the default channel layer setting *Default* should be used. Otherwise, define one of the four channels *Voice*, *Video*, *Text* or *Web*. For your convenience, use the two combined settings *Voice/Video* or *Text/Web* in case one Output item is to be active in those combinations. Using a combined channel is the same as defining two Output items with the individual channel setting, with all other settings being the same.
Language	To identify the language in which the respective Output item is provided. If an Output item is appropriate in all relevant languages, the default language layer setting *Default* should be used. For languages that are used in multiple countries, both the specific language options (e.g. *English (SG), English (US)*) and the generic option (*English*) are provided.
Occurrence	Enables you to play different outputs depending on how often the Output object has already been visited. *Always* means that regardless of the number of previous visits, the respective Output item is always active. *Only once* means that the Output item is active only the first time the Output object is visited. And *If >= N* means that the Output item is active only if this is at least the Nth time that the Output object has been visited.
Input mode	To tag the Output item with an input mode, so that different outputs can be played depending on the current state of the input mode system layer. This is typically used for applications that are designed to be controlled with both input modes: *Voice* and/or *DTMF*. Using this setting you can, for example, define one Output item that is played when the input mode is set to *Voice* (*Please say yes or no*), and a different one for input mode *DTMF* (*Please press 1 for yes and 0 for no*), both in the same Output object. For more information on designing input mode-specific applications, refer to Chapter 9 – *How to Use Different Input Modes* in the *Design Guide*. You can set the input mode either to *Voice*, *DTMF*, or *Voice+DTMF*. If you do not want to distinguish your Output items with regard to input mode, leave the setting at *Default*. In this case, the output item is applicable to any input mode. The initial state of the input mode system layer can be set in the Service object. For more details, refer to Configuring a Service in Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Barge-in	Enables barge-in by selecting *True* or disables it by selecting *False*. Barge-in is a feature in the voice and video channel that allows callers to interrupt an output message and respond without waiting until the output has finished. The feature itself is essentially provided by the media platform whereas VoiceObjects provides the possibility to switch barge-in on or off. In the text and Web channel, the barge-in setting is ignored. The default setting *Default* indicates that the barge-in setting from the current dialog context is taken. The initial value of the barge-in can be set in the Service object. For more details, refer to Configuring a Service in Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*. **Note:** When using Output objects within other Output objects, the barge-in setting of the innermost output is applied to each sub-portion of the total prompt.

The actual output is defined in the **Prompt** field.

Prompt

An output can be defined by entering plain text in the text field or by inserting new or existing Output, Variable, Collection, Expression, Script, Audio, Video, Text, Silence, and Layer objects.

Note: Audio, Video and Silence objects are only applicable in the voice and video channel. In the text and Web channel, these objects will be ignored. The textual channels offer a special character for display purposes, though, which is ignored in the voice and video channel. Use the pipe symbol "|" if you want to include a line break in the output. More than one line break can be achieved by using more pipes, separated by a blank. A double-pipe with no blanks "||" will result in a pipe symbol to be displayed. In the Web channel, you can also include XHTML elements in the Output items. Use elements like `<b>`, `<i>`, `<center>` etc. to modify the layout of the output, or use `<div>` or `<span>` to make use of style sheets (CSS). See HTML below for more information on this.

Existing objects like Variable or other Output objects can be referenced in a prompt by typing in their names using the shortcut notation described in Chapter 6 – *Object Editors* in the *Desktop for Eclipse Guide* and *Desktop for Web Guide*, respectively.

Note: A Collection object will be presented cell by cell, ignoring the XML tags. If a Variable or Script object contains XML code which is formatted as a collection, it will be treated the same way. Any other XML code will be presented including the tags.

Copyright © 2001-2011 Voxeo Germany GmbH

Random Prompting

The Output object supports true random prompting, a valuable technique for good User Interface design for voice and video applications. In the text and Web channel, random prompting is less frequently used.
Random outputs can be achieved by simply providing multiple Output items with the same language layer, custom layer (defined in the **Layer** field), occurrence level, channel, and input mode setting.

The algorithm that determines which Output item gets played is as follows:

1. Start with all Output items defined within the Output object.
2. Evaluate the custom layer conditions of all Output items and eliminate those whose layer condition evaluates to false.
3. Eliminate all Output items that do not have their language layer set to either the currently active language, a super-set thereof (e.g. *English* vs. *English (US)*), or *Default*.
4. Eliminate all Output items that do not have their input mode layer set to either the currently active input mode, or *Default*.
5. Eliminate all Output items that do not have their channel layer set to either the currently active channel, or *Default*.
6. Assuming this is the Nth visit to the Output object, eliminate all Output items that have the occurrence level set to *>=(N+1)* or higher. For *N>1*, eliminate all Output items with occurrence level *Only once*. If an Output item with occurrence level *>=N* exists, eliminate all those with setting *>=(N-1)* or lower. If no Output item with occurrence level *>=N* exists, use the one with the highest number below N and eliminate all others below.
   Example: An Output item that is tagged with *Only once* is only considered for playing the very first time the Output object is visited. An Output item that is tagged with *If >= 2* is considered for playing upon the second visit to the Output object, and for all subsequent visits unless an item with tag *If >= 3* is defined in the same Output object.
7. If more than one Output item remains, select one at random from among the remaining set. True randomizing ensures that all available prompts are played before a repetition occurs.
8. If no Output item remains, do nothing.

Note: Even if no Output item remains, the Output object's postprocessing (if defined) will still be processed.

If an Output object is used in reprompting, its occurrence settings are interpreted differently. All items with settings *Always* or *Only once* are ignored. For all items with settings *If >= N* the value N is interpreted as the reprompting level.

For a detailed description of how to use the reprompting mechanism, refer to Chapter 8 – *Advanced Event Handling* in the *Design Guide*.

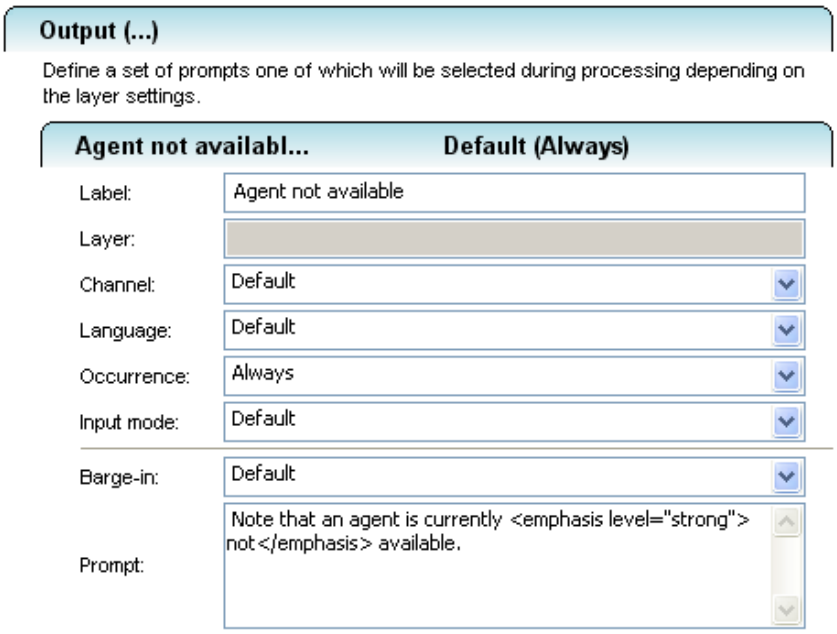
SSML

The Speech Synthesis Markup Language (SSML), developed by the Voice Browser Working Group of W3C, allows adding meta information to text that is supposed to be read out through a speech synthesis system in voice or video applications. By using SSML you have better control over how the TTS engine synthesizes speech. It allows you to define voice-related parameters like volume, speed, pitch, emphasis,

Copyright © 2001-2011 Voxeo Germany GmbH

pronunciation, etc. For more information on SSML, refer to the specification at the W3C Website, or to chapter 4.1.1 of the VoiceXML 2.0 specification.

SSML markup can be included in any Output item as well as alternative text items within the Audio object. VoiceObjects Server passes the extra code to the media platform without changes. The following example shows how to use SSML to tell the TTS engine to put emphasis on a specific word within a given prompt:



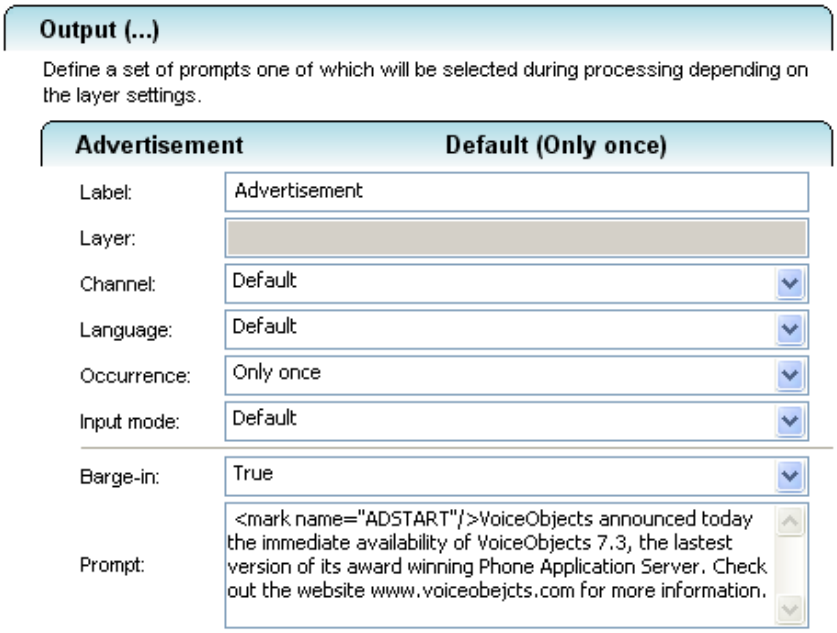
Caution: Since the markup you enclose in the Output item definition is passed to the platform as it is, make sure to provide the correct syntax required by SSML. Note that it is up to the platform or TTS engine, respectively, if and how SSML is interpreted. If a TTS engine does not support SSML, it might decide to read the XML tags out aloud.

Prompt Marker

SSML allows you to embed markers between different prompts within an output. The prompts can be both TTS and prerecorded audio. The markers can be used to detect if the caller barged in on a prompt, as well as the position within a prompt at which this barge-in occurred. The system function LASTRESULT(type, [index], [phase]) with *type=markname* returns the name of the last marker processed by the media platform. With *type=marktime* you get the time in milliseconds that has passed since processing the last marker. LASTRESULT is available through the Expression object.

Markers can be defined within Output items by putting a `<mark>` element with a *name* attribute at the desired position. In the following example, a marker "ADSTART" is defined at the beginning of an advertisement:

Copyright © 2001-2011 Voxeo Germany GmbH



In this example, the LASTRESULT function with *type=marktime* can help in finding out if the message was heard entirely by the caller or if he or she barged in too early during the advertisement. The latter could, for instance, result in a reprompt.

HTML

The Web channel works on the basis of (X)HTML, which is rendered at call time by the server. You can enhance the rendered markup by embedding HTML tags in Output items.

Examples:

Element	Description
<b>	Prints the text **bold**.
<i>	Prints the text in *italics*.
<u>	Underlines the text.
** **	Centers the text in the display.
<h1>, <h2>, …	Applies a heading layout to the text.
<font size="…">	Applies given size to the text.
<font color="…">	Shows text in given color.
<img src="…">	Embeds an image in the display using the given URL.

While these tags are only relevant in the Web channel, the content itself (=prompts) is usually the same in the text and Web channel. By default, the server therefore removes all XML tags from an Output definition within the text channel. This enables developers to use the same Output item for both channels, setting the channel layer to

Copyright © 2001-2011 Voxeo Germany GmbH

Text/Web. Note, though, that a string like “XX_
YY”, where “” denotes blanks, will result in two blanks “XX__YY”.

By default, Web browsers apply their own styles when it comes to displaying the content on the screen. As a Web developer, you can influence the layout by using *cascading style sheets* (CSS). VoiceObjects Server uses three default style sheets for all pages rendered for a dialog, depending on the driver selected. For the *Mobile Web XHTML 1.0* driver, the file is called *default.css*. For the *Apple iPhone Web XHTML 1.0* driver, the file is called *iphone.css*. For the *Rich Web Client XHTML 1.0* driver, the file is called *richweb.css*. Find them in your VoiceObjects installation at *Resources\System\StyleSheet*. You can either modify this CSS file, or reference your own by using the corresponding tuning parameter *Presentation – Style Sheet URL*. When using this parameter, the default style sheet will not be applied.

For all elements rendered by the server that are relevant for layout, different *classes* are used, so you can modify the output of these elements by defining styles for these classes. The following table lists all available classes:

Element + Class	Used for
div.voOutput	Prompt in Output object
a.voMenuItem	Menu items in Menu object
a.voInputItem	Items in Input object, if tuning parameter *Presentation – Input* is set to *menu*
a.voConfirmOutput	Confirm output in Confirmation object
a.voCorrectionItem	Correction items in Confirmation object
a.voListNavigationItem	Navigation commands in List object
a.voListSelectionItem	Selection commands in List object
a.voNavigationItem	Single custom and standard navigation command
div.voNavigation	Entire block of navigation commands
input.voSubmit	“Proceed” button
div.voSubmit	“Proceed” button; use this to be able to center the button
input.voText	Text input field of Input object, if tuning parameter *Presentation – Input* is set to *text*
input.voPassword	Password input field of Input object, if tuning parameter *Presentation – Input* is set to *password*
input.voRadio	Radio buttons of Input object, if tuning parameter *Presentation – Input* is set to *radio*
option.voOption	Options (drop-down list) of Input object, if tuning parameter *Presentation – Input* is set to *list*
img.voTopLogo	Top logo image, defined through tuning parameter *Presentation – Top Logo URL*
img.voBottomLogo	Bottom logo image, defined through tuning parameter *Presentation – Bottom Logo URL*

Copyright © 2001-2011 Voxeo Germany GmbH

Element + Class	Used for
h6.voTitle	Phone Simulator – Web only. The <title> tag is converted into <h6 class="voTitle"> as the first child element of <body>, so that it is shown within the phone screen.

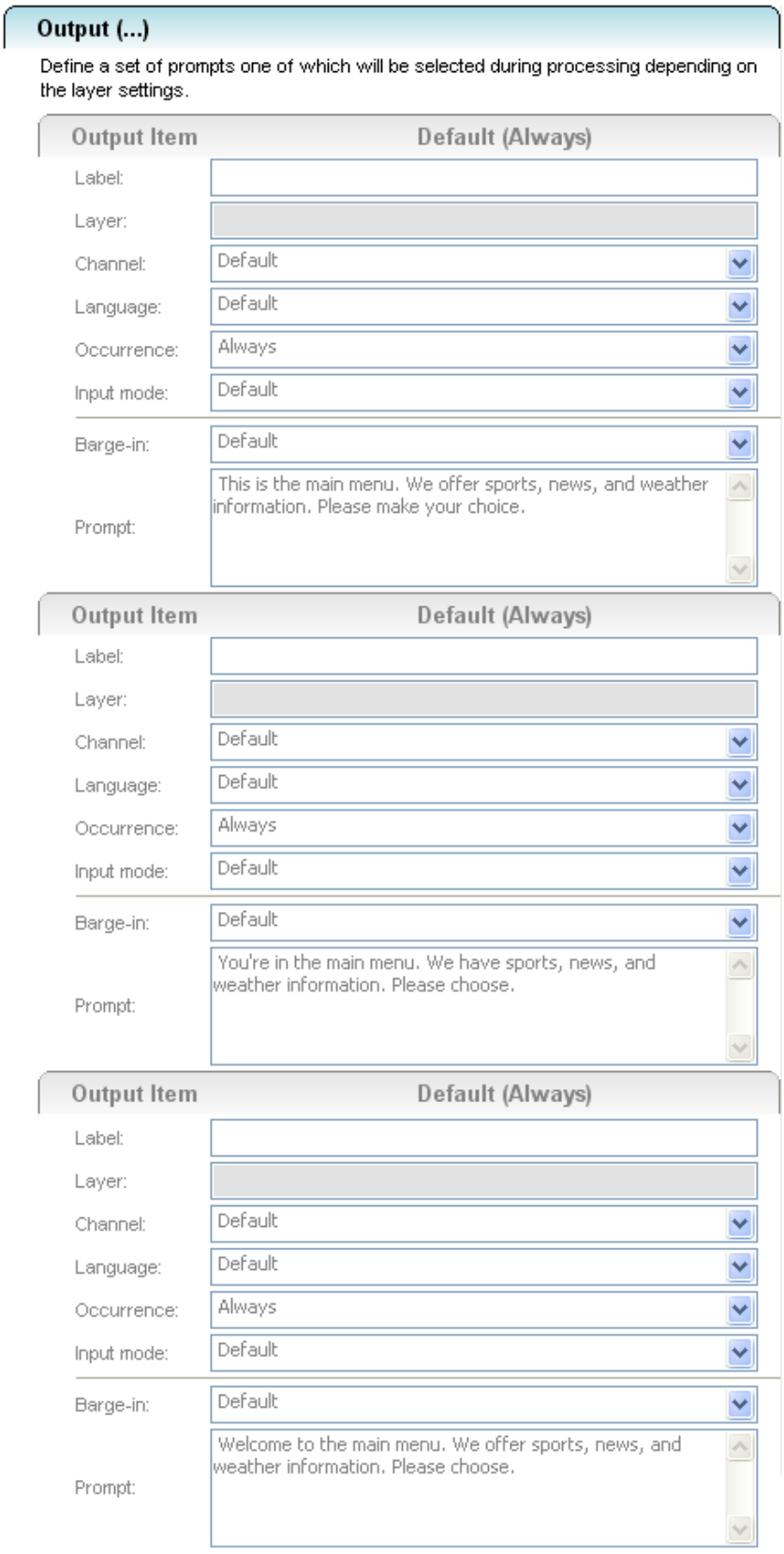
In addition to *class*, the server adds the *id* attribute to all elements relevant for layout. This attribute consists of the type and name of the object that this page was rendered for, with the syntax *type_name*. Type and name are converted to lower-case and all special characters such as blanks, underscores or hyphens are removed from the name. Using this attribute, the style sheet can define formatting that only applies to specific objects of an application.

Usage Examples

Random prompting

The first example shows random prompting. By providing three different Output items with the same language setting and without any additional layers (see below), the server will randomly select one of them at call time.

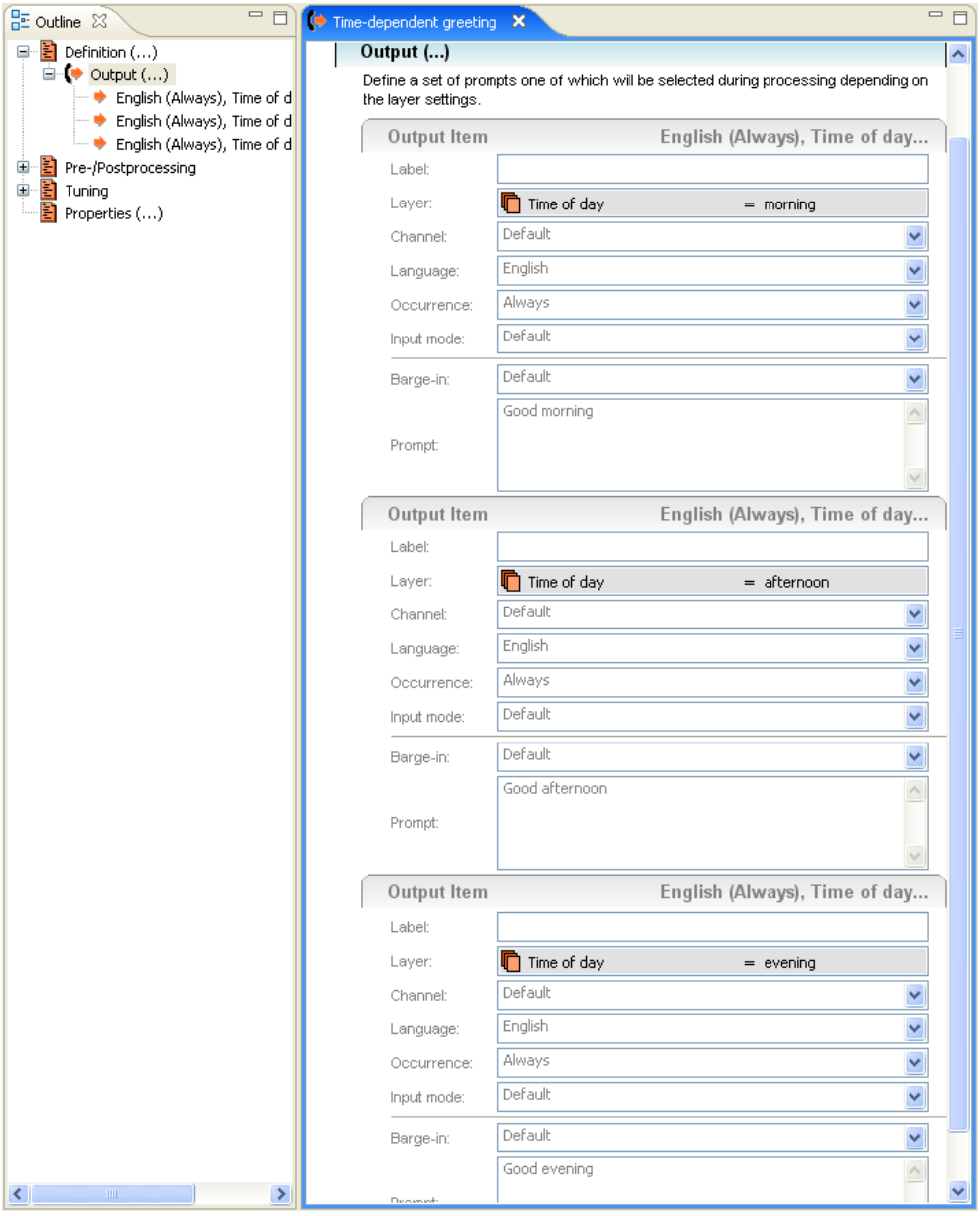
Copyright © 2001-2011 Voxeo Germany GmbH



Copyright © 2001-2011 Voxeo Germany GmbH

Custom layers

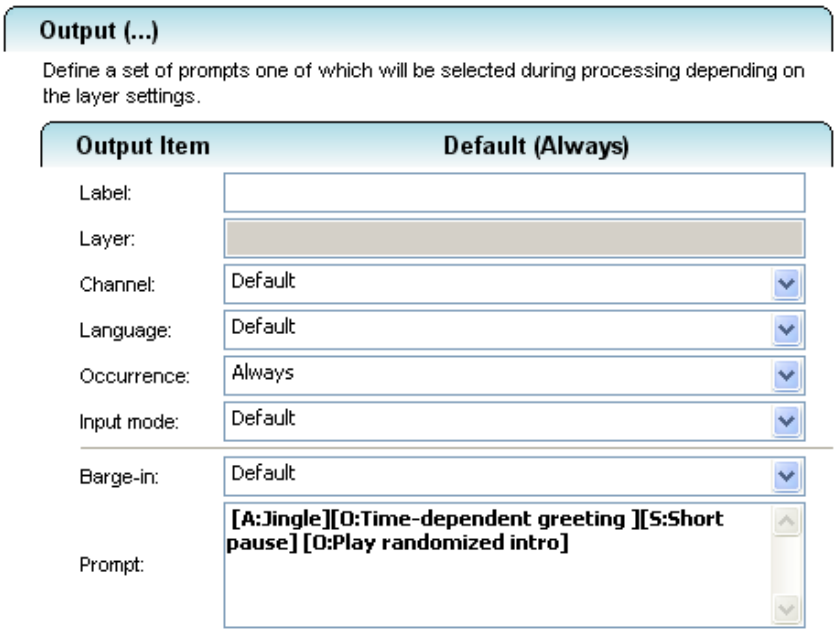
By using custom layers associated with the Output items, it is easy to build time-dependent greetings, for instance. Create one Output item for each time of day as shown below, and associate a layer with each of them, testing whether it is morning, afternoon, or evening.



Copyright © 2001-2011 Voxeo Germany GmbH

Lively welcome outputs

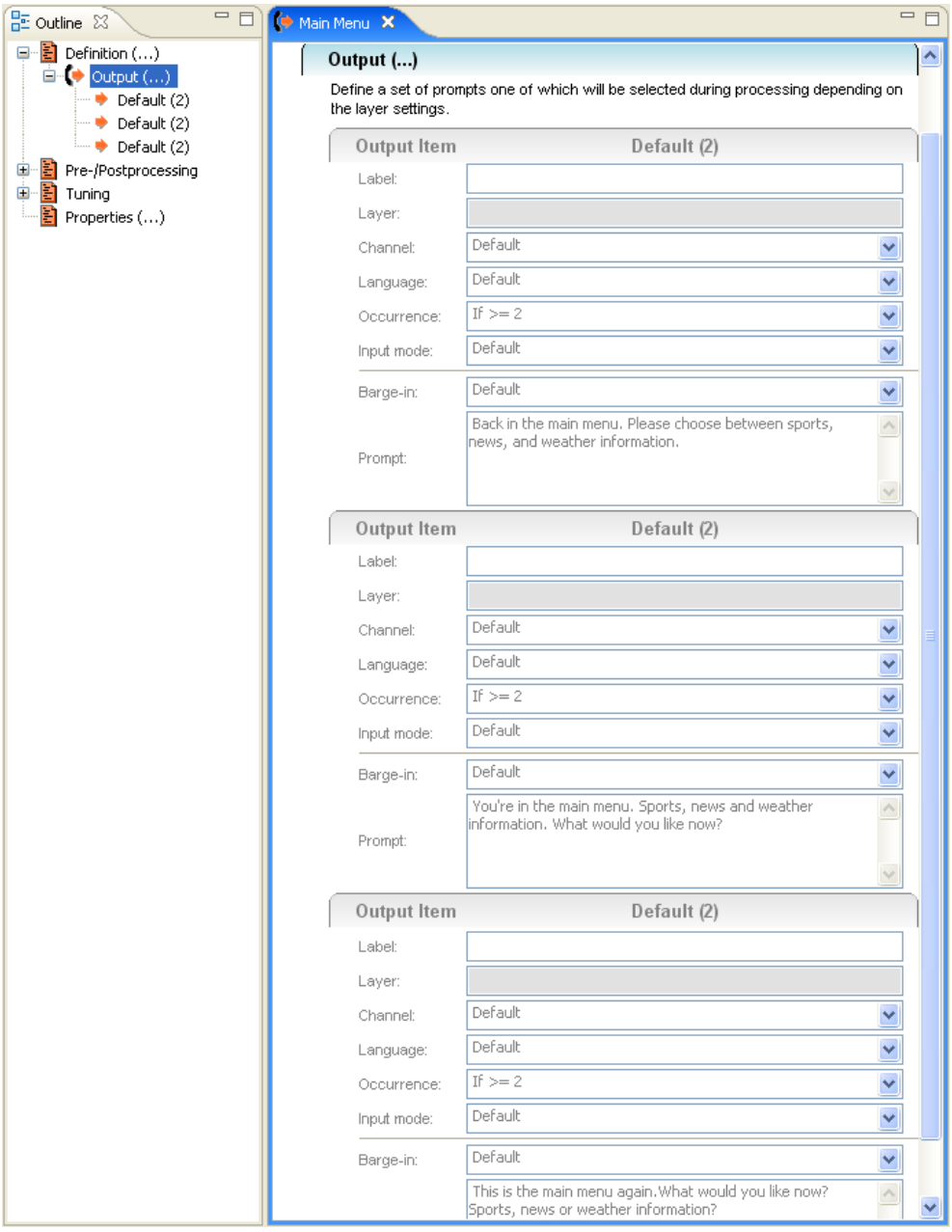
By combining an audio jingle, the time-dependent greeting shown above, and a randomized intro as shown in the first example, a varied and lively welcome output for an application can be created with minimal effort. Note that the Output item in the following example only consists of references to other objects, such as Audio, Output, and Silence.



Occurrence level

In many cases, it is desirable to have different outputs depending on whether a menu is reached the first time or subsequently. This is easily achieved using the occurrence level setting as shown below. Note how this can be combined with random prompting, to let the menu sound fresh, i.e. non-robotic, even when the caller returns to it many times during the call.

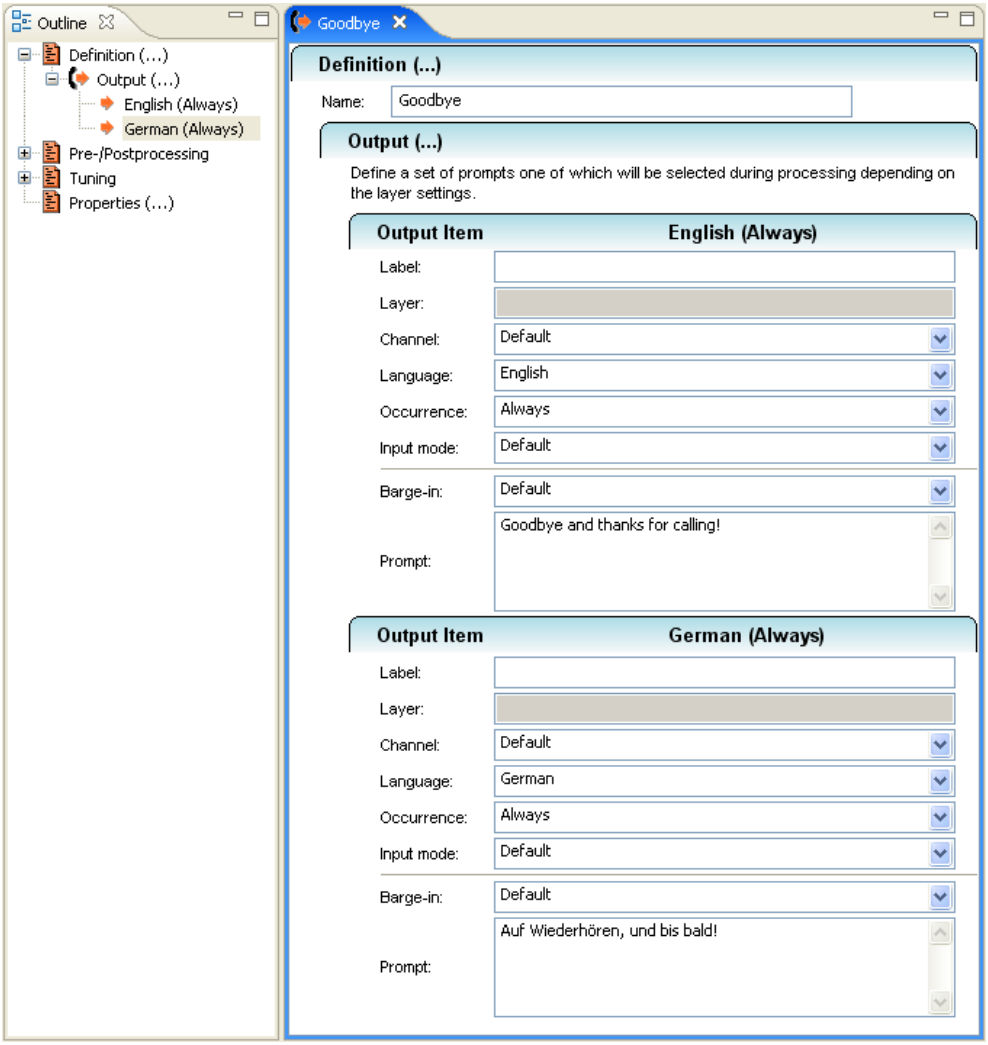
Copyright © 2001-2011 Voxeo Germany GmbH



Language layer

The final example shows the use of the language layer for a multilingual goodbye prompt in *English* and *German*.

Copyright © 2001-2011 Voxeo Germany GmbH



Output objects within other Output objects

Highly sophisticated prompts can be designed by using Output objects within other Output objects. For example, a prompt *Welcome [Name]!* can be split into two Output objects to provide a fully dynamic greeting. The table below describes the object definitions, where each row of condition and content describes a single Output item.

Output object	Condition	Content
Welcome	time = morning age > 30	Good morning
	time = afternoon age > 30	Welcome
	time = evening age > 30	Good evening
	time = morning, afternoon, evening age <=30	Hi

Copyright © 2001-2011 Voxeo Germany GmbH

Output object	Condition	Content
	time = morning, afternoon, evening age <= 30	Hello
Welcome complete		[O:Welcome] [V:Name]

Processing the Output object *Welcome complete* during a call will first process the *Welcome* Output object with all its conditions.

For a 36 year old caller named *Henry*, the output will be *Good Morning Henry*, *Welcome Henry* or *Good evening Henry* depending on the time of day.

For a 23 year old caller named *Lisa*, the output will be one of the two random selections *Hi Lisa* or *Hello Lisa*.

Advanced usage example: Local barge-in

Barge-in, i.e. the capability for the caller to interrupt system output, is routinely used in state-of-the-art voice applications. Its flexibility however is often rather limited on the current generation of media platforms. For example, it is often difficult to define a specific range of prompts over which barge-in should extend.

Consider the following example: A voice application that provides the download of mobile phone ring tones has the following initial dialog flow:

Object – Caller	Dialog Flow
	*Welcome to Chart Ring. We let you download the Top Ten tunes to your phone!*
	*Here's how it works…* (Plays explanatory intro)
	Plays the ring tone of the day
	*How about it? Download this ring tone now?*

The caller may interrupt the application during the intro if he already knows how to use it. In this case, though, barge-in should not skip all prompts up to and including the question *Download this ring tone now?* Instead it would be preferable to still present the ring tone of the day and to follow up with the question of whether the caller wants to buy it. We call this *local barge-in*, since it allows you to define the scope of the barge-in and limit it to a certain set of prompts. Note that this is very different from simply switching barge-in off, which would force the caller to listen to certain prompts in their entirety before being able to say anything.

Local barge-in can be achieved as follows:
The prompt (or prompts) to show the behavior should be used within the initial output of an Input object instead of within an autonomous Output object. As the grammar to be used in the Input, define a simple TTG utterance that is unlikely to be entered by the caller (e.g. *this is unlikely to be entered*). Both options *Enable auto-advance on No Input* and *Enable auto-advance on No Match* need to be selected. In the **Event Handling** section, define a handler for the *Caller Help* event and set continuation to *Proceed*, if desired. Finally, the tuning parameter *Input – Speech Timeout* should be set to *0.01 seconds*.

Copyright © 2001-2011 Voxeo Germany GmbH

The modified dialog flow looks as follows:

Object – Caller	Dialog Flow
	*Welcome to Chart Ring. We let you download the Top Ten tunes to your phone! Here's how it works…* (Plays explanatory intro; uses local barge-in)
	Plays the ring tone of the day
	*How about it? Download this ring tone now?*

If the caller barges in during the initial prompt, this is interpreted as an input. Since the input is unlikely to match the grammar, this will cause a No Match event. Due to the setting *Enable auto-advance on No Match* the application proceeds to the next object in the dialog flow and plays the ring tone of the day.

If, on the other hand, the caller does not interrupt the initial prompt then the application waits for an input. Since the *Input – Speech Timeout* is set to a very small value, a No Input event is thrown almost immediately. Due to the setting *Enable auto-advance on No Input* the application proceeds to the next object in the dialog flow and plays the ring tone of the day.

VoiceObjectsXML Definition

The Output object is represented by the VoiceObjectsXML element `<output>`. It has one attribute and five groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<output>` element uses the embedded `<outputItem>` element.

Output

Attributes

- **`type`**
  Defines whether this Output object is to be used as either *initial* or *reprompt*. If not specified, the type defaults to *initial*. When an Output object is linked, this attribute may be overwritten.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Output object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Output object.
- **`+<outputItem>`**
  Defines the list of available Output items.
- **`<tuning>`**
  Defines the tuning settings for the Output object.

Copyright © 2001-2011 Voxeo Germany GmbH

- **`<sequence usage="postprocessing">`**
  Defines the postprocessing sequence for the Output object.

Example

```
<output>
  <outputItem language="en-US">
    <text>Welcome to Prime Insurance.</text>
  </outputItem>
  <outputItem language="de-DE">
    <text>Herzlich willkommen bei Prime Insurance.</text>
  </outputItem>
</output>
```

OutputItem

Attributes

- **`label`**
  A text string providing a name for the Output item.
- **`bargein`**
  Defines the barge-in behavior of the Output item. Can be *true*, *false*, or *default*. If not specified, defaults to *default*.
- **`language`**
  Defines the language for which this Output item is valid. Can be *default* or a valid language code (e.g. *de-DE*, *en-US*, etc.). If not specified, defaults to *default*. Appendix A – Language Codes contains a list of all language codes available in VoiceObjects together with the respective language they represent.
- **`occurrence`**
  Defines the occurrence level for which this Output item is valid. Can be *always*, *once*, or an integer between 1 and 10. If not specified, defaults to *always*.
- **`inputMode`**
  Defines the input mode for which this Output item is valid. Can be *default*, *voice*, *dtmf*, or *voicedtmf*. If not specified, defaults to *default*.
- **`channel`**
  Defines the channel(s) for which this Output item is valid. Can be *default*, *voice*, *video*, *text*, *web*, *voiceVideo*, or *textWeb*. If not specified, defaults to *default*.
- **`layer`**
  Defines the layer for this Output item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form "Layer=State" or "Layer!=State" where "State" is the label of a state for the layer "Layer".

Note: When using Output objects within other Output objects, only the barge-in setting of the outermost Output object is used.

Children

- **+(`<audio>`, `<video>`, `<silence>`, `<output>`, `<layer>`, `<variable>`, `<expression>`, `<collection>`, `<script>`, `<text>`)**

Copyright © 2001-2011 Voxeo Germany GmbH

Defines the actual output as a concatenation of various elements.
Note that a single blank will automatically be inserted between any two consecutive non-text children during import into the VoiceObjects Metadata Repository or deployment to VoiceObjects Server in order to ensure proper output behavior.

Examples

```
<outputItem>
  <text>The annual premium for your </text>
  <variable link="#Car Manufacturer"/>
  <variable link="#Car model"/>
  <text> starts at just </text>
  <variable link="#Premium"/>
  <text> dollars.</text>
</outputItem>
```

Object Interoperability

The following table contains all object types that can reference an Output object:

Icon	Object Name	Use Case Example
	Module	An Output object can be used within the embedded sequence of a Module object, as well as be referenced for the welcome and goodbye output of the Module object.
	Input	An Output object can be referenced within an Input object.
	Output	An Output object can be referenced from within other Output objects.
	Sequence	Most commonly, an Output object is used within a Sequence object in a dialog flow.
	Menu	An Output object can be referenced within each Menu item in a Menu object.
	Confirmation	An Output object can be referenced in multiple places inside a Confirmation object, as well as used as the destination object within a Correction item.
	List	An Output object can be referenced in multiple places inside a List object.
	If	An Output object can be used within either the THEN item or the ELSE item of an If object.
	Case	An Output object can be used in any WHEN item of a Case object.
	Loop	An Output object can be used within the embedded sequence of a Loop object.
	Goto	An Output object can be referenced via a Goto object.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Object Name	Use Case Example
	Hyperlink	An Output object can be the destination of a Hyperlink object.
	Recording	An Output object can be referenced within a Recording object.
	Transfer	An Output object can be referenced within a Transfer object.
	Pause	An Output object can be referenced as both the welcome and the pause output of a Pause object.
	Exit	An Output object can be referenced within an Exit object.
	OSDM	An Output object can be linked in the parameter set within any OSDM object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of an Output object should indicate which type of prompt is played to the caller. The table below lists three examples:

Name	Description
 Play insurance fee	Plays the insurance fee retrieved from a database to the caller.
 Randomized intro	Plays a randomized intro to the Insurance Portal, selecting from 5 different prompts.
 Time-dependent greeting	Plays a greeting that depends on the current time of day, e.g. *Good morning* or *Good evening*.

Copyright © 2001-2011 Voxeo Germany GmbH

Pause

Overview

The Pause object enables the caller to suspend an application at any point during the call. When processing a Pause object the regular dialog flow will be temporarily paused and the caller can say a configurable wake-up command (e.g. *Continue*) to return to the original dialog flow. The last processed object in which the pause was initiated will be repeated.

The Pause object is particularly useful for applications used, for example, in the car, where situations may arise that require the caller's full attention, or if the caller needs to search for a number or PIN to log into an application. The application can then be paused by the caller and later re-activated when the caller is ready to proceed.

Due to its nature the Pause object is only motivated for the voice and video channel; in the text and Web channel it will not be processed. For further information on channels refer to Chapter 10 – *How to Support Multiple Phone Channels* in the *Design Guide*.

The Object Definition below covers the configuration of the Pause object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Pause object belongs to the object category Actions.

Dialog Flow Scenario

The first dialog flow example presents a pause dialog within a voice portal in which the caller activates a Pause object through a hyperlink while listening to the introduction of the main menu. An intermittent beep tone followed by three seconds of silence is played back during the pause to signal to the caller that the call is still active. After the caller says the wake-up command *wake-up* the dialog continues with the main menu prompt.

Object – Caller	Dialog Flow
	[Music jingle] [*Hello and welcome to the Voice Portal.*]
	Main menu! You have the following options …
Caller	*Pause*
	The dialog will be paused now. You can say wake-up or you can press any key on your phone to continue the application.
	[Beep] [Silence of 3 seconds] [Beep] [Silence of 3 seconds] … [Beep] [Silence of 3 seconds]
Caller	*Wake-up*

Copyright © 2001-2011 Voxeo Germany GmbH

Object – Caller	Dialog Flow
	*Main menu! You have the following options …*

The second dialog flow example is slightly different in that the repeated Output during the pause plays a music track followed by a TTS prompt. This provides the caller with a hint on how to end the Pause object and return to the original dialog flow. Also, a different wake-up command is used here (*proceed* instead of *wake-up*).

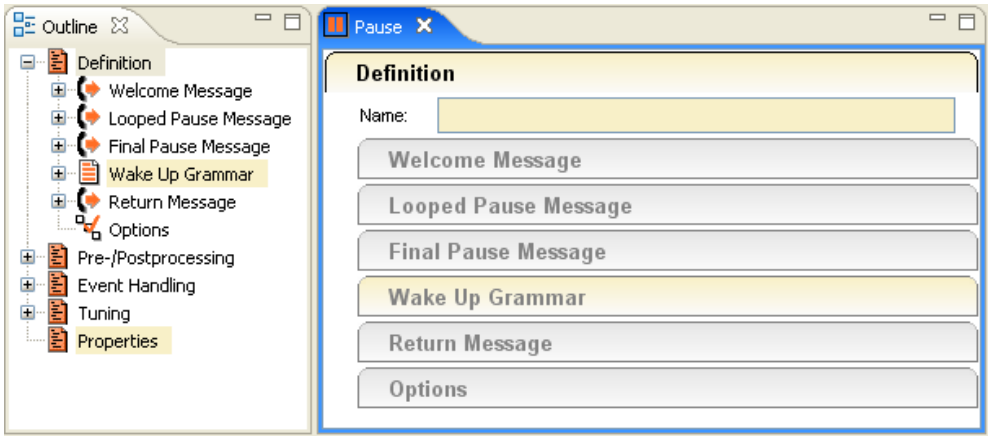
Object – Caller	Dialog Flow
	[Music jingle] [*Hello and welcome to the Voice Portal.*]
	*Main menu! You have the following options …*
 Caller	*Pause*
	*The dialog will be paused now. You can say "proceed" to continue the application.*
	[Music track] *Just say "proceed" to continue.* [Music track] *Just say "proceed" to continue.* … [Music track] *Just say "proceed" to continue.*
 Caller	*Proceed*
	*You're back in the main menu! You can choose between …*

Object Definition

The **Definition** of the Pause object provides the following sections:

- Welcome Message
  To define a welcome message that notifies the caller that the pause has been activated.
- Looped Pause Message
  To define an output to be played during the pause. It signals to the caller that the call is still active.
- Final Pause Message
  To define an output to be played after the maximum number of reiterations (if limited) has been reached and before the redirect object is processed (if defined).
- Wake Up Grammar
  To specify the utterances which will terminate the pause.
- Return Message
  To define a return output to notify the caller that the application is being resumed.
- Options
  To configure processing options.

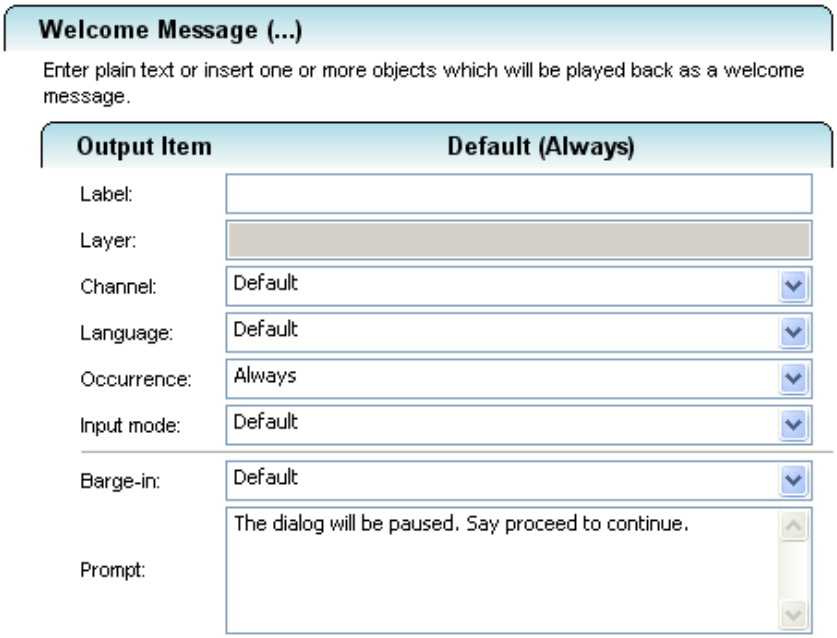
Copyright © 2001-2011 Voxeo Germany GmbH



For information regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Tuning*, and *Properties* in this *Object Reference*.

Welcome Message

In the **Welcome Message** section, optional output can be defined that is played when the Pause object has been activated. It is typically used to tell the caller which command will terminate the pause and return to the original dialog flow.

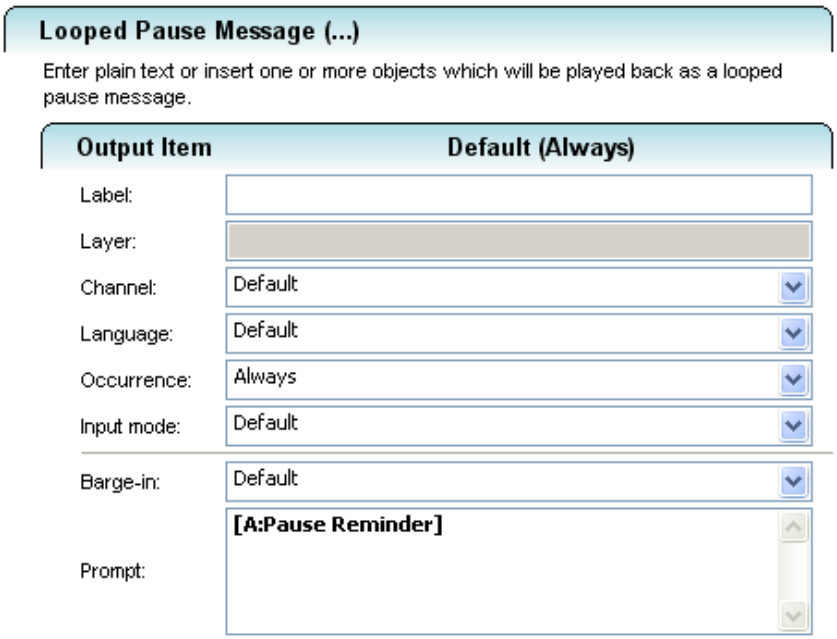


For more details on output definitions refer to the Output object in this *Object Reference*.

Looped Pause Message

In the **Looped Pause Message** section, optional pause output can be specified that is played repeatedly during the processing of the Pause object, to signal to the caller that the call is still active and is listening for a specified wake up command.

Copyright © 2001-2011 Voxeo Germany GmbH



For more details on output definitions refer to the Output object in this *Object Reference*.

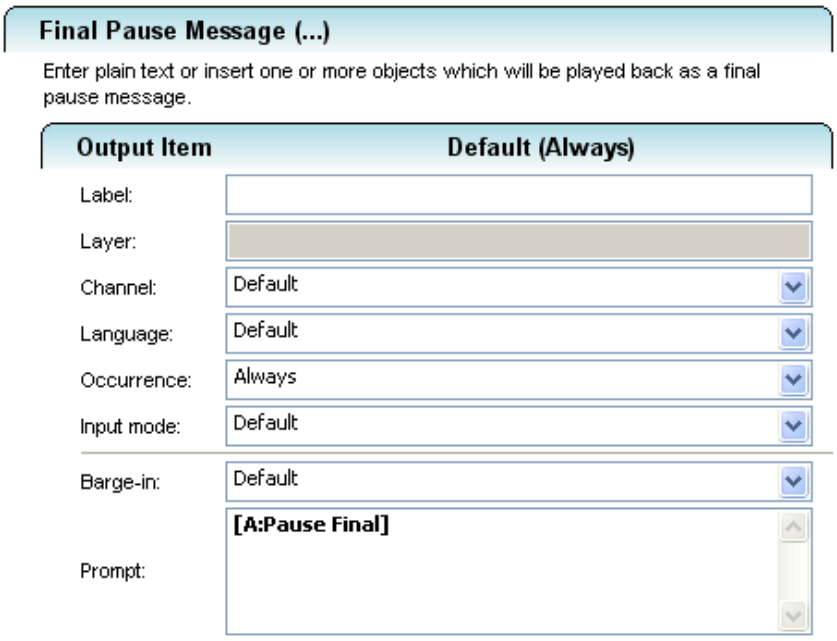
Within the pause output a prerecorded prompt, a music track, a beep tone, or a TTS message can be played. In many cases it is useful to add a Silence object to the output definition (see the first dialog flow scenario above with the beep tone signal).

Another way of producing a silence phase at the end of each pause output iteration is to set the speech timeout parameter of the media platform to the desired interval (e.g. 5 seconds) in the Tuning definition. Note, that if no explicit timeout property is set in the application, the default (typically 3 seconds) is taken.

Final Pause Message

If the number of reiterations is limited (see Options), the final pause message, if defined, is played after the maximum number of reiterations has been reached. If a redirect object is defined (see Options), this becomes the next object to be processed. If it is not defined, the dialog proceeds with the next object in the dialog flow.

Copyright © 2001-2011 Voxeo Germany GmbH

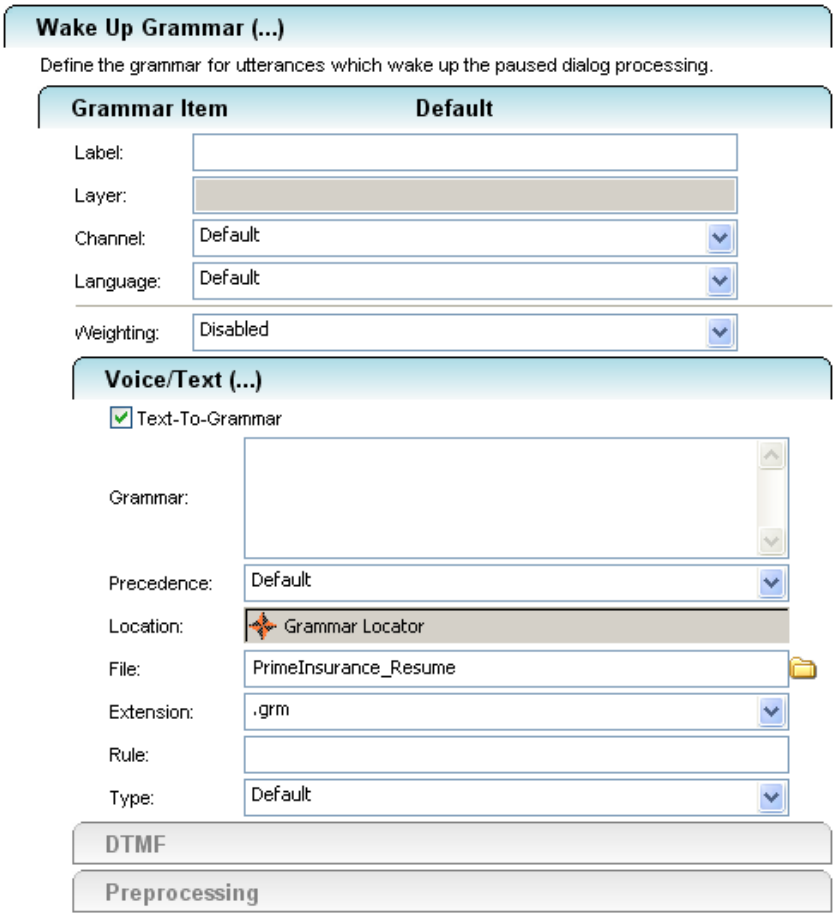


For more details on output definitions refer to the Output object in this *Object Reference*.

Wake Up Grammar

The wake up grammar defines the utterances the caller can say and/or the DTMF keys the caller can press to reactivate the application. This grammar does not need to fill a slot (though it may).

Copyright © 2001-2011 Voxeo Germany GmbH



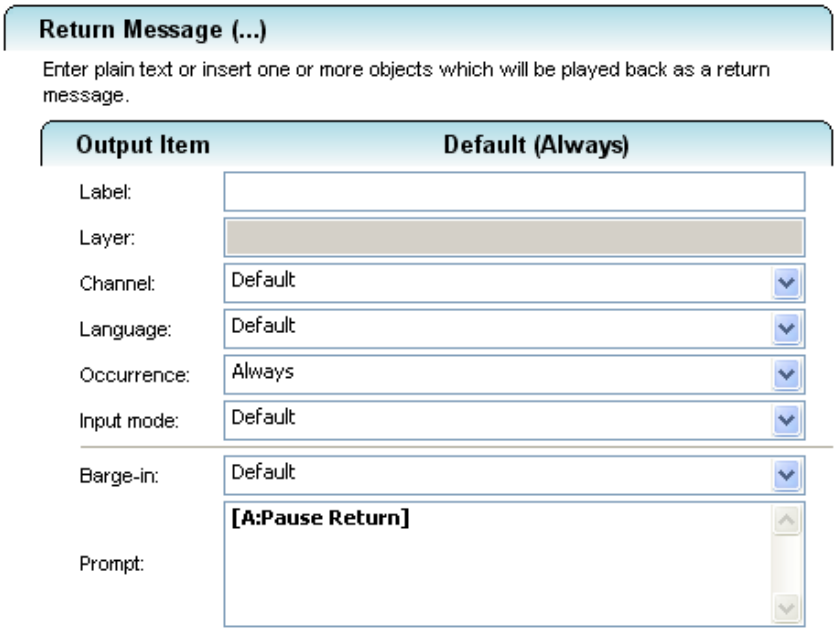
For more details on grammar definition refer to the Grammar object in this *Object Reference*.

Note: If standard navigation is defined for the command *skip*, the application can also be resumed by using this command. For more information on standard navigation, see Navigation in this *Object Reference*.

Return Message

In the **Return Message** section, optional return output can be specified that is played before the pause ends and the dialog flow continues, starting with the next object in the dialog flow.

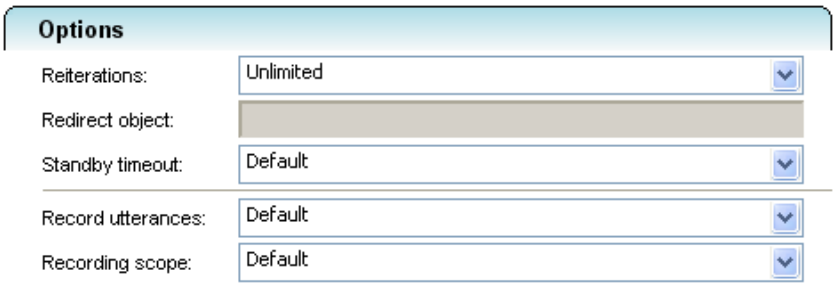
Copyright © 2001-2011 Voxeo Germany GmbH



For more details on output definitions refer to the Output object in this *Object Reference*.

Options

In the **Options** section, the **Reiterations** field is used to limit the number of iterations of the looped pause message. The default value is *unlimited*, in which case the dialog will not resume until the caller explicitly uses the wake up command. Setting it to a number [n] means that, if the caller says nothing, the application will stay in the pause for at most [n] iterations, and then wake up automatically.



The limitation can be provided as a constant value or assigned during call time through an integer value of a Variable, Expression, Script, or Layer object.

If a **Redirect object** is defined, this becomes the object that is processed after reaching the maximum number of reiterations. Note, that the server does not return to the original dialog flow if a redirect object is defined. If the dialog is supposed to continue normally after processing the redirect object, you need to make sure that the necessary objects are part of the redirect object, which in this case would most likely be a Sequence object.

Tip: The duration of one iteration approximately equals prompt duration + speech timeout. Use this formula to calculate the number of iterations necessary in case you do not want the caller to remain inside the Pause object for more than a certain period

Copyright © 2001-2011 Voxeo Germany GmbH

of time. Note, though, that a caller barge-in (which is not the wake up command) interrupts the looped pause message, shortening the length of the current iteration. This can be avoided by setting the tuning parameter *Barge-in Type* to *hotword* (if supported by the media platform).

The **Standby timeout** field is used to specify the maximum time the caller may remain in the pause before the dialog session will be removed. Instead of selecting one of the values from the drop-down list, it is possible to type in a custom value. This value is interpreted as seconds, e.g. for a custom timeout of 5.5 minutes, type in 330. The default setting *Default* indicates that this timeout is defined within the Service object. For more details on services, refer to *Configuring Servers and Services* in Chapter 2 – in the *Deployment Guide*.

Note: The standby timeout does not end the call on the media platform. It only finishes the current dialog session on the server. If the caller activates the wake up command after this timeout has elapsed, it will not be possible to return to the dialog but an error message is played instead, generated by VoiceObjects Server.

Record utterances specifies whether utterances made in this object are to be recorded (*True*) or not (*False*), given that utterance recording is switched on for this session. *Always* indicates that utterances in this object are always recorded when utterance recording is enabled on the service, regardless of statistical filtering. By default the value is *Default*, indicating that the current value of the corresponding dialog context setting RECORDUTTERANCES is used.

Recording scope defines the scope in which recordings will be made. If set to *Recognition*, only those utterances will be recorded that were successfully recognized (including the activation of hyperlinks). If set to *No Match*, only those utterances will be recorded that triggered a No Match event. If set to *All*, any recording will be stored. If set to *Default*, the current value of the corresponding dialog context setting RECORDINGSCOPE is used. By default the value is *Default*.

Due to the nature of recordings, the feature of recording utterances is only available in the voice and video channel. For more information on this refer to Chapter 6 – *Recording of Utterances* in the *Deployment Guide*.

Note: There are specific limitations within the Pause object when using it within a Transfer object performing a bridged call transfer or within a Recording object performing a recording session. Neither of these sessions can be paused and then continued at a later point in time. When activating a pause during an active bridged transfer or an active recording session via a speech or DTMF triggered Hyperlink object, the hyperlink activation will terminate the transfer or the recording session and the dialog flow will continue with the processing of the Pause object. Nevertheless when the pause is finished and the original dialog flow continues, the corresponding Transfer or Recording object will restart.

Not all media platforms certified by VoiceObjects support the activation of a hyperlink during a bridged call transfer or a recording session. In this case the pausing of the dialog during these sessions is not possible.

VoiceObjectsXML Definition

The Pause object is represented by the VoiceObjectsXML element `<pause>`. It has five attributes and ten groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

Pause

Attributes

- **`iterations`**
  Defines the maximum number of iterations spent within the pause. An integer value >= 0. The default is 0, which indicates an unlimited number of iterations.
- **`redirectObject`**
  Defines an object (through a reference) that is processed when the maximum number of iterations is exceeded and the caller did not exit the pause via a wake-up command.
- **`standbyTimeout`**
  Defines the standby timeout for the Pause object. Either *default* or a numerical value that is interpreted as seconds. If not specified, defaults to *default*.
  May be static text or a reference to an Expression, Script, or Variable object.
- **`recordUtterances`**
  Defines whether to use utterance recording. One of *true*, *false*, *always*, *default*. If not specified, defaults to *default*.
- **`recordingScope`**
  Defines the scope for utterance recording. One of *all*, *noMatch*, *recognition*, *default*. If not specified, defaults to *default*.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Pause object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Pause object.
- **`<output usage="welcome">`**
  Defines the welcome message.
- **`<output usage="pause">`**
  Defines the looped pause message.
- **`<output usage="final">`**
  Defines the final pause message.
- **`<output usage="return">`**
  Defines the return message.
- **`<grammar>`** [required]
  Defines the wake-up grammar.
- **`<eventHandling>`**
  Defines the event handling for the Pause object.
- **`<tuning>`**
  Defines the tuning settings for the Pause object.
- **`<sequence usage="postprocessing">`**
  Defines the postprocessing sequence for the Pause object.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<pause>
  <output usage=”welcome”>
    <outputItem><text>The application has been
paused.</text></outputItem>
  </output>
  <output usage=”pause”>
    <outputItem><audio link=”#Pause beep”/></outputItem>
  </output>
  <grammar>
    <grammarItem>
      <grammarDefinition mode=”voice” ttg=”true”>
        continue
      </grammarDefinition>
    </grammarItem>
  </grammar>
</pause>
```

Object Interoperability

The following table contains all object types that can reference a Pause object:

Icon	Object Name	Use Case Example
	Sequence	A Pause object can be used within a Sequence object to automatically pause the dialog at a certain point.
	Menu	A Pause object can be initiated within a Menu object.
	If	A Pause object can be either the THEN or the ELSE item of an If object.
	Case	A Pause object can be used in any WHEN item of a Case object to pause the dialog under a certain condition.
	Goto	The Pause object can be referenced within a Goto object.
	Hyperlink	A Pause object can be used as the target of a Hyperlink object. This is the most common use case. A Pause object gets activated on demand by the caller through a voice command (see dialog flow scenario above).

Copyright © 2001-2011 Voxeo Germany GmbH

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Pause object should indicate the way this pause is used. The table below lists two examples:

Name	Description
Application pause	Temporarily pauses the call; activated via a Hyperlink object
Optional Pause at start	Call pause before the application starts.

Copyright © 2001-2011 Voxeo Germany GmbH

Plug-In

Overview

The Plug-In object enables the use of custom VoiceXML code. It can be used to access proprietary extensions available on a media platform (e.g. for call control or billing purposes), as well as to embed existing VoiceXML applications into the VoiceObjects environment.

The Plug-In object is only supported in the voice and video channel; it will be ignored in the text and Web channel.

The Object Definition below covers the configuration of the Plug-In object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Plug-In object belongs to the object category Resources.

Dialog Flow Scenario

The dialog flow example shows the use of a Plug-In object for speaker verification in a banking application. This example assumes that the underlying media platform offers a custom tag to perform speaker verification.

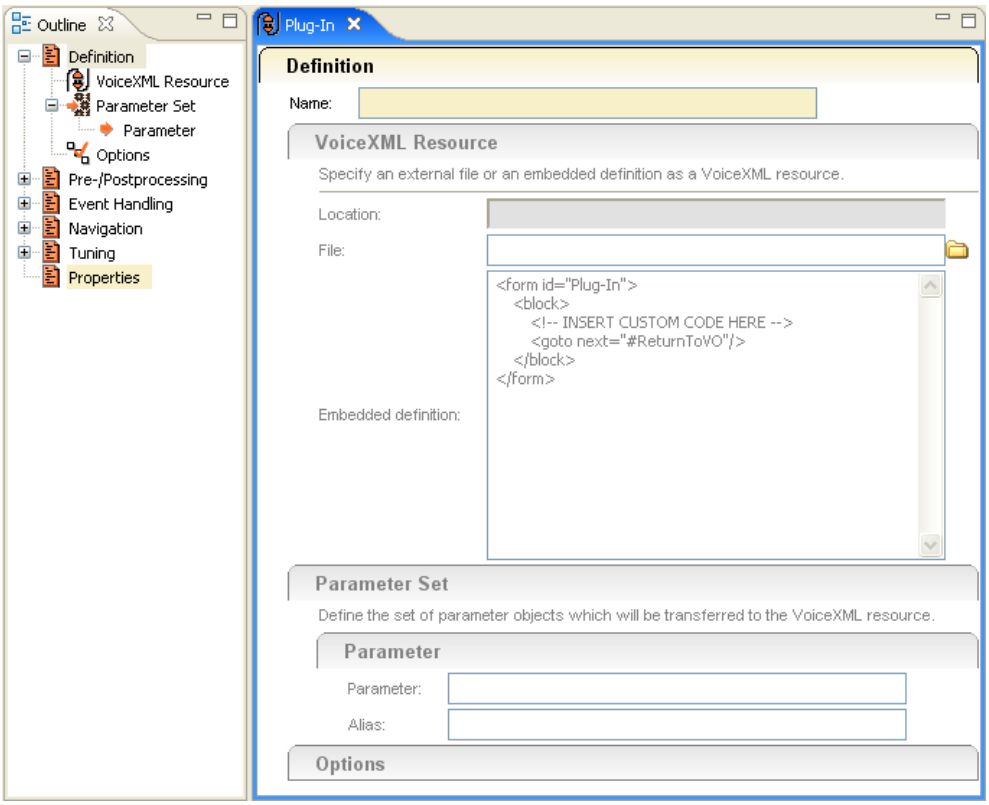
Object – Caller	Dialog Flow
	*To log into your account, please say the following identification phrase.*
	Plug-In object containing custom tag that verifies the caller and provides the result back in a variable.
	Decide whether caller was successfully verified, and continue dialog accordingly.

Object Definition

The **Definition** of the Plug-In object provides the following sections:

- VoiceXML Resource
  To define the VoiceXML code either as embedded text or as a reference to an external file.
- Parameter Set
  To specify the set of parameters to be exchanged with the VoiceXML code.
- Options
  To specify processing options for the Plug-In object.

Copyright © 2001-2011 Voxeo Germany GmbH



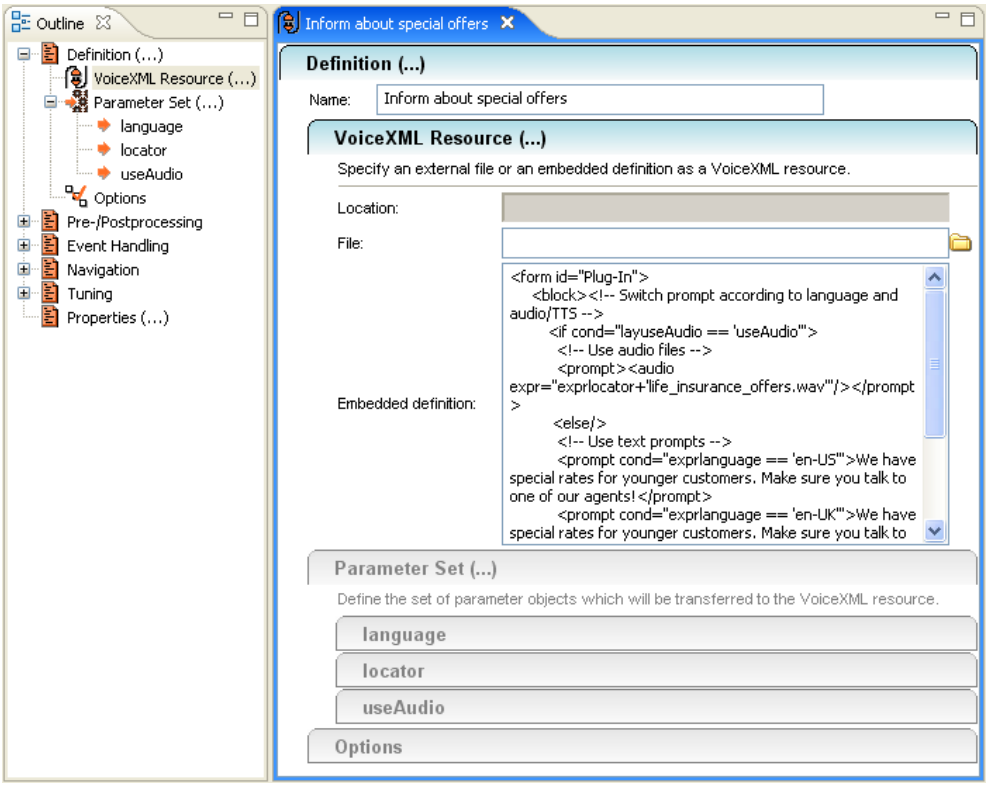
For information regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Navigation*, *Tuning*, and *Properties* in this *Object Reference*.

VoiceXML Resource

The **VoiceXML Resource** section defines the VoiceXML code to be executed within the Plug-In object. It can either be provided as an embedded definition typed into the **Embedded Definition** field (as shown in the example below), or via a reference to a file. In the latter case, an optional Resource Locator object and the name of the file need to be provided. The file may also be a server-side script that dynamically creates the VoiceXML code at call time.

Note: If both an embedded definition and a file reference are given, the embedded definition takes precedence.

Copyright © 2001-2011 Voxeo Germany GmbH



The VoiceXML resource can also be defined dynamically by using a Variable, Expression, Layer, or Script object.

There is an important difference between using an embedded and using an external definition of the VoiceXML resource. When using an embedded definition, VoiceObjects Server automatically creates wrapping VoiceXML code containing whatever navigation, event handling, and tuning is defined either locally within the Plug-In object or inherited from higher-level objects. Using embedded definitions, all elements are allowed that can be placed below the `<vxml>` element, especially `<form>` and `<menu>`.

When using an external definition, no wrapping VoiceXML is created. Instead, the external VoiceXML file is used exactly as it is provided. Any navigation, event handling, etc. needs to be manually coded within the VoiceXML code.

Note: If you make use of namespaces inside embedded VoiceXML code, make sure to declare the namespace through the *xmlns* attribute, e.g. in the `<form>` element.

To return to the server from within the custom VoiceXML code, there are two options, one of which is only available when using an embedded definition:

1. When using an embedded VoiceXML resource definition, the server automatically creates an additional `<form>` that provides a return statement. It can be accessed via the VoiceXML tag

   ```
   <goto next="#ReturnToVO"/>
   ```

   When using this return mechanism, all variables that were passed into the Plug-In object (see Parameter Set) are also automatically submitted back, and the corresponding objects are updated in the server.

Copyright © 2001-2011 Voxeo Germany GmbH

2. In both embedded and external definition, the return can be achieved by a custom `<submit>` tag within the VoiceXML code. This tag must be of the following form:

   ```
   <submit expr="returnURL" namelist="dialogID …"/>
   ```

   where the `namelist` must contain at least the parameter `dialogID` which is used to identify the appropriate dialog session to be continued within the server. In addition, the `namelist` may contain whichever variables from the VoiceXML code are to be passed back into the server. Both `returnURL` and `dialogID` are passed into the VoiceXML code automatically, so they can be used without having to explicitly include them in the Parameter Set.

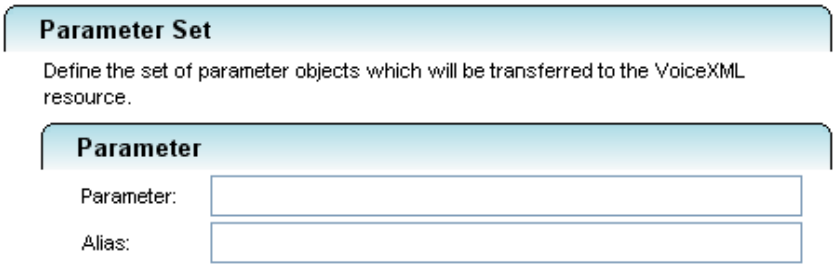
   Even if you have an external file to be processed (either static VoiceXML or some CGI script), it is nevertheless recommended to use an embedded definition within the Plug-In object, as this gives you full control over the passing of parameters to the script and passing them back to the server. In most cases, it is appropriate to use a `<submit>` element and include `dialogID` and `returnURL` in the `namelist` attribute.

If the VoiceXML code fails for some reason (e.g. a syntax error) it depends entirely on the media platform whether this can be handled within VoiceObjects Server or not. When using an embedded VoiceXML resource definition, it is advisable to specify an event handling for the event type *Error – Media Platform*. This creates a handler for the VoiceXML event `error` within the VoiceXML wrapper that the server places around the custom code. Whether this handler is activated depends on the behavior of the respective media platform.

Note: If your code contains an input state definition, e.g. through a `<menu>` or `<field>` element, you need to set the **Contains input state** option, otherwise the correct processing in terms of standard navigation, Infostore logging and utterance recording cannot be guaranteed. See Options for more details.

Parameter Set

The **Parameter Set** section defines an optional list of parameters to enable the exchange of processing information with the VoiceXML code.



Each **Parameter** section provides the following properties:

Property	Description
Parameter	Specifies the value of the parameter. Possible parameter types are Variable, Collection, Expression, Layer, Resource Locator or Script objects, and constants. When using a Resource Locator object as a parameter, its URL definition is passed to the script. The field can be left empty to denote an empty value.

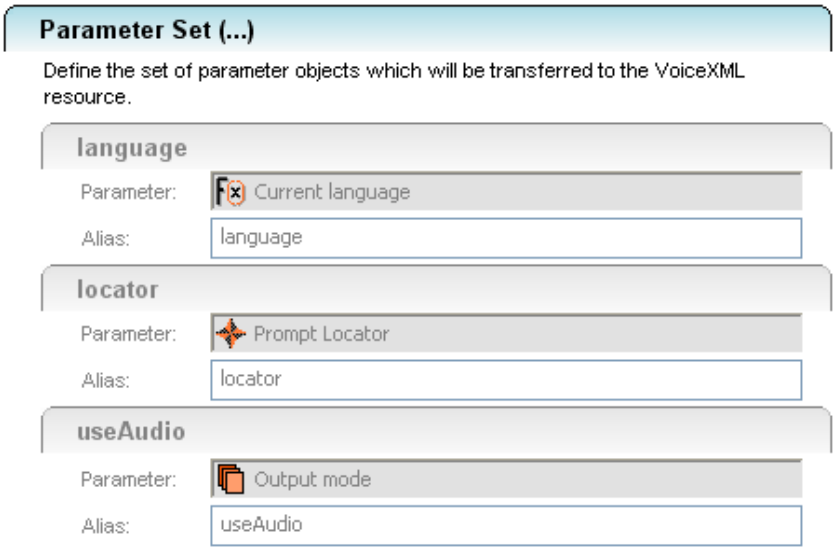
Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Alias	Optionally specifies the name under which the parameter is accessible within the VoiceXML code. If left empty, the reference ID of the object defined in the **Parameter** field will be used as the name. This means in particular that for constant values an alias needs to be defined (as they do not have a reference ID). Constant parameters without a defined alias are ignored.

Note that neither the entries in the **Alias** fields nor the reference IDs which you pass into the plug-in may contain special characters such as hyphens.

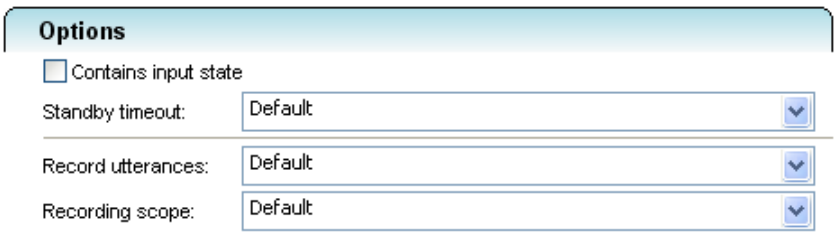
For parameters passed to the custom VoiceXML code, the prefix *var* is attached to the reference ID (or alias, if defined) of all Variable objects and constants, the prefix *lay* is attached to the reference ID (or alias, if defined) of all Layer objects, and the prefix *expr* is attached to the reference ID (or alias, if defined) of all Collection, Expression, Script, and Resource Locator objects. When accessing the variables within the VoiceXML code or passing them back in a manual `<submit>` as described above, the names must be provided using these same prefixes. Note that only variables, collections, and layers can reasonably be passed back to the server.

Any number of parameters can be exchanged with the VoiceXML code.



Options

The **Options** section of the Plug-In object specifies four processing options.



Copyright © 2001-2011 Voxeo Germany GmbH

The check box **Contains input state** tells the server if the VoiceXML code also defines an input state, e.g. contains a `<menu>`, `<field>` or `<record>` element. This instructs the server to consider the plug-in for standard navigation. If the check box is clear, the plug-in is skipped when navigating over it using commands like *go back* or *forward*. If it is selected, the server stops at the plug-in and executes it as a full-fledged dialog step.
In addition, this property controls whether the plug-in will occur as an input state in the Infostore table VOLDDSSEQ. If it is selected, the plug-in will be logged as an input state, otherwise it will not. If you use the *ReturnToVO* form to go back to the server (see the section on VoiceXMLResource above) the transition type (DS_TR_TYPE) for the plug-in will be *Recognition* by default. If you want to avoid this, you cannot use the *ReturnToVO* form but instead need to submit manually to the server, optionally calling the *count* script with the desired event name that will control the transition type to be logged.

Standby timeout indicates how long the VoiceObjects dialog session is kept alive while the call is being handled by the plug-in code. If the call stays in the plug-in for longer than the standby timeout, a return to the VoiceObjects dialog session is not possible.
When using a Variable, Expression, Script, or Layer object to define the standby timeout at call time, or when typing in a custom value instead of selecting one of the values from the drop-down list, the value needs to be a positive number and will be applied as seconds, e.g. for a custom timeout of 5.5 minutes, type in 330. The default setting is *Default*, indicating that the standby timeout setting is controlled by the service. For more information see Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*. Alternatively, a standby timeout can be selected from the drop-down list or it can be provided via a Variable, Expression, Script, or Layer object.

Record utterances specifies whether utterances made in this object are to be recorded (*True*) or not (*False*), given that utterance recording is switched on for this session. *Always* indicates that utterances in this object are always recorded when utterance recording is enabled on the service, regardless of statistical filtering. By default the value is *Default*, indicating that the current value of the corresponding dialog context setting RECORDUTTERANCES is used.

Note: Utterance recording will be turned off if the check box **Contains input state** is not selected, even if the **Record utterances** option is *true* at call time. This is due to the fact that if the plug-in code does not contain an input state, no recording can be made.

Recording scope defines the scope in which recordings will be made. If set to *Recognition*, only those utterances will be recorded that were successfully recognized (including the activation of hyperlinks). If set to *No Match*, only those utterances will be recorded that triggered a No Match event. If set to *All*, any recording will be stored. If set to *Default*, the current value of the corresponding dialog context setting RECORDINGSCOPE is used. By default the value is *Default*.

Note: The server can only store recordings of utterances made within code segments rendered by the server, e.g. inherited No Match handlers or hyperlinks. If you want to record utterances within your own VoiceXML, see Plug-in object in Chapter 6 – *Recording of Utterances* in the *Deployment Guide* for a description of the code you need to use.

Copyright © 2001-2011 Voxeo Germany GmbH

VoiceObjectsXML Definition

The Plug-In object is represented by the VoiceObjectsXML element `<plugin>`. It has seven attributes and eight groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<plugin>` element uses the embedded `<value>` and **`<parameterSet>`** elements.

Plug-In

Attributes

- **`location`**
  Defines the location from where the VoiceXML file is to be retrieved. Must be a reference to a Resource Locator object.
- **`file`**
  The file name of the VoiceXML file. Can be a constant name, or a reference to an Expression, Script, or Variable object.
- **`embeddedRef`**
  Defines the plug-in code through a reference to an Expression, Script, or Variable object.
- **`standbyTimeout`**
  Defines the standby timeout for the plug-in. Either *default* or a numerical value that is interpreted as seconds. If not specified, defaults to *default*.
  May be static text or a reference to an Expression, Script, or Variable object.
- **`inputState`**
  Indicates whether the code that is processed within the plug-in contains an input state that should be considered during standard navigation. Legal values are *true* or *false*. If not specified, defaults to *false*.
- **`recordUtterances`**
  Defines whether to use utterance recording. One of *true*, *false*, *always*, *default*. If not specified, defaults to *default*.
- **`recordingScope`**
  Defines the scope for utterance recording. One of *all*, *noMatch*, *recognition*, *default*. If not specified, defaults to *default*.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Plug-In object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Plug-In object.
- **`<value>`**
  Optional embedded constant definition of the plug-in code.
- **`<parameterSet>`**
  Defines the parameter set for the Plug-In object.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<eventHandling>**
  Defines the event handling for the Plug-In object.
- **<customNavigation>**
  Defines the custom navigation settings for the Plug-In object.
- **<tuning>**
  Defines the tuning settings for the Plug-In object.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the Plug-In object.

Example

```
<plugin location="#Plugin locator" file="playAd.vxml">
  <parameterSet>
    <item alias="useAudio" object="#Audio enabled"/>
    <item alias="audioPath" object="#Audio locator"/>
  </parameterSet>
</plugin>
```

Value

Children

- **CDATA**
  Embedded code for the Plug-In object.

Example

```
<value>
  <![CDATA[
    <form>
      <block>
        <prompt>Hello world!</prompt>
      </block>
    </form>
  ]]>
</value>
```

ParameterSet

Children

- **+<item>**
  Defines the list of entries in the parameter set. The use of the *alias* attribute is optional when the *object* attribute is used, and required when the *value* attribute is used.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<parameterSet>
  <item alias=”customer” object=”#Customer number”/>
  <item alias=”locale” value=”de-DE”/>
</parameterSet>
```

Item

Attributes

- **`object`**
  Defines a reference to an object. If this attribute is defined, the `value` attribute must not be defined.
- **`value`**
  Defines a constant value. If this attribute is defined, the `object` attribute must not be defined.
- **`alias`**
  Defines the parameter alias by which the parameter can be accessed within the VoiceXML code.

Example

```
<item alias=”premium” object=”#Annual premium”/>


<item alias=”locale” value=”en-US”/>
```

Object Interoperability

The following table contains all object types that can reference a Plug-In object:

Icon	Object Name	Use Case Example
	Module	A Plug-In object can be used within the embedded sequence of a Module object.
	Sequence	Most commonly, a Plug-In object is used within a Sequence object in a dialog flow.
	Menu	A Plug-In object can be used as the destination object within a Menu item.
	Confirmation	A Plug-In object can be used as the destination object within a correction item inside a Confirmation object.
	If	A Plug-In object can be used within either the THEN or the ELSE item of an If object.
	Case	A Plug-In object can be used in any WHEN item of a Case object.
	Loop	A Plug-In object can be used within the embedded sequence of a Loop object.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Object Name	Use Case Example
	Goto	A Plug-In object can be referenced via a Goto object.
	Hyperlink	A Plug-In object can be the destination of a Hyperlink object.
	Exit	A Plug-In object can be the destination of an Exit object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Plug-In object should indicate which task it performs. The short description should contain a brief explanation of what is done within the plug-in, and which data is exchanged with it. The table below lists three examples:

Name	Description
Verify caller	Uses an `<object>` tag to authenticate the caller using speaker verification.
Invoke legacy application	Invoke the VoiceXML-based legacy application. After it is finished, the service continues.
Extended agent transfer	Transfer the call to an agent, simultaneously dropping all information gathered so far into the Call Center information system for agent pop-up.

Copyright © 2001-2011 Voxeo Germany GmbH

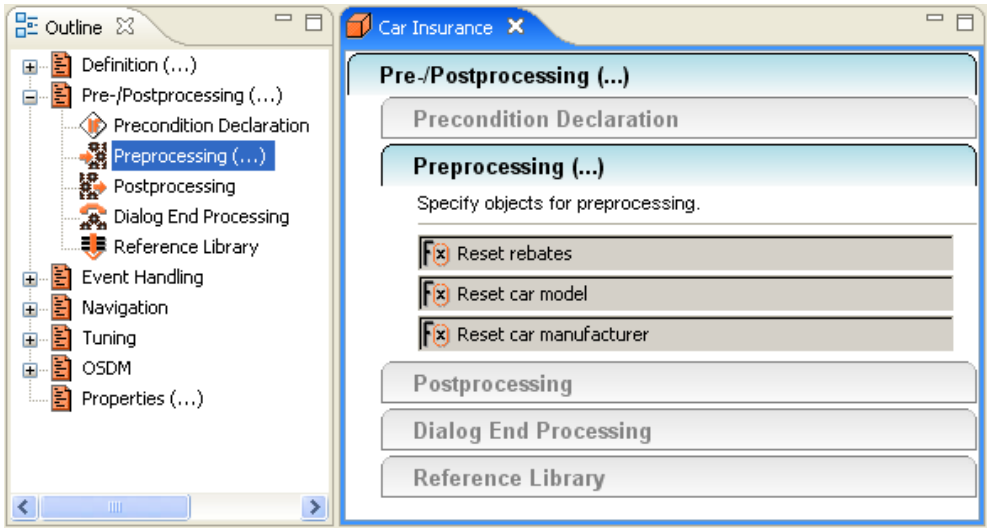
Pre-/Postprocessing

Overview

Pre- and postprocessing can be used to define single objects or sequences of objects to be processed before or after a certain object, respectively. The advantage is that whenever the object is used in the dialog flow, its embedded pre- and postprocessing objects are always automatically processed along with it. The effect is the same as if the preprocessing objects were put in the dialog flow right before the affected object, and the postprocessing objects right after it.

Note: In objects like Menu or Confirmation, the postprocessing objects will not be processed if, after selecting an item, the dialog flow never returns to the original object, e.g. due to a hyperlink or a Goto object being processed.

Pre- and postprocessing includes the definition of preconditions evaluated before processing an object (see below).



Defining a Precondition

In the **Precondition Declaration** section a precondition can be defined, that is evaluated before processing an object. The object is only processed if its precondition evaluates to *true*. Preconditions can be defined either by specifying an embedded expression, or by linking an Expression, Variable, Collection, or Script object. In this case, the precondition is *true* unless the return value of the linked object is an empty string, 0 (zero) or FALSE (in any combination of upper and lower case letters).

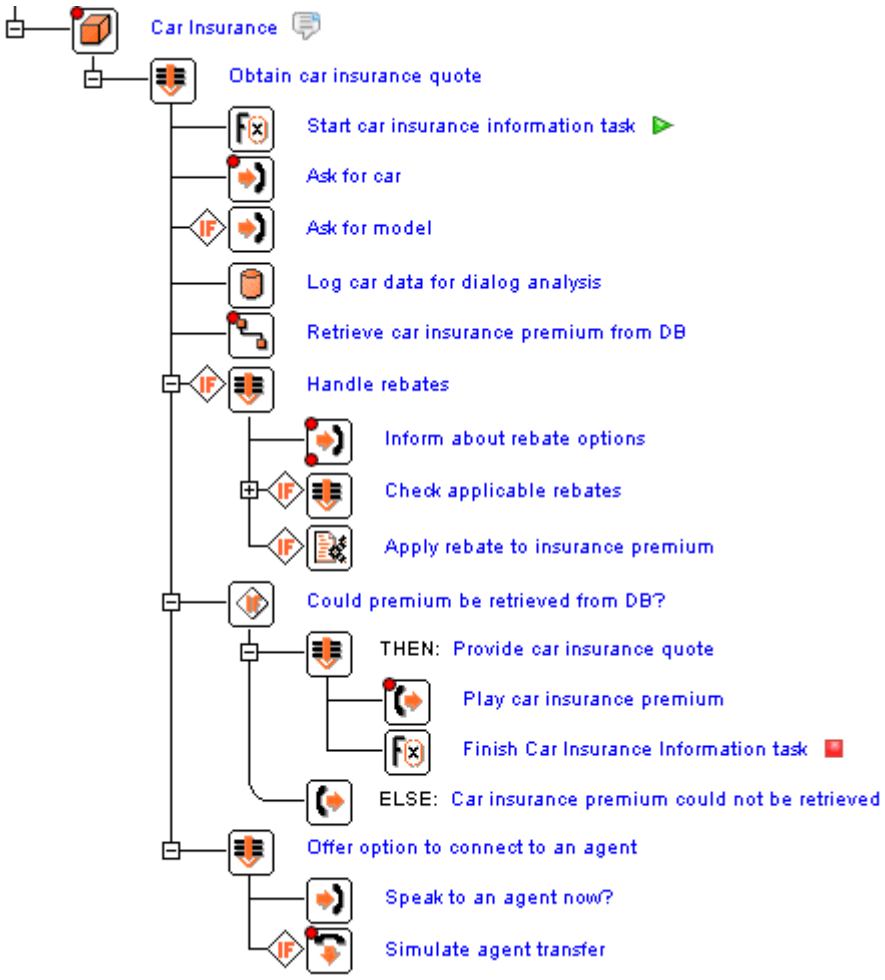
You can also use a layer condition as a precondition. In this case, select *Layer condition* as the function and link the layer condition as first argument. For more details on this, refer to the Expression object in this *Object Reference*.

By using preconditions dialog flows can be designed to allow dynamic object processing during call time. As an example, an Input object might be processed only in case the information it asks for is not yet available.

The following example is part of the *Prime Insurance* application. In the *Car Insurance* module, the caller is asked for the make and model of his or her car. The caller can either respond with the make only, or both make and model. The precondition of the

Copyright © 2001-2011 Voxeo Germany GmbH

Ask for model Input object checks if the content of the *Car Model* Variable object is empty. Only then does the Input object get processed. If the caller responded with the make and model, the *Ask for model* Input object is skipped.



For more information on the expression functions and on the definition of a precondition expression, see the Expression object in this *Object Reference*.

Defining Pre-/Postprocessing

In the **Pre-/Postprocessing** section (represented by a tab in Desktop for Web), single objects or sequences of objects can be specified to be processed before or after the object itself. This feature may be used, for example, to initialize parameters used within the object or to apply operations on the result provided by the object.

To specify objects to be processed *before* a particular object, use the **Preprocessing** section.

For objects to be processed *after* a particular object use the **Postprocessing** section.

In the **Dialog Designer**, objects with a pre- or postprocessing definition are indicated by a red dot at the top left (for preprocessing) or bottom left (for postprocessing) corner of its icon (see also Chapter 5 – *Dialog Designer* in the *Desktop for Eclipse Guide* or *Desktop for Web Guide*).

Copyright © 2001-2011 Voxeo Germany GmbH

The actual sequence of objects specified can be displayed in the **Dialog Designer** by a setting in the display options, as described in Chapter 7 – *Basic Commands* in the *Desktop for Eclipse Guide or Desktop for Web Guide*.

VoiceObjectsXML Definition

Pre-/postprocessing

Pre- and postprocessing objects are defined using `<sequence usage=”preprocessing”>` and `<sequence usage=”postprocessing”>` children, respectively.

As an example for postprocessing, see the following excerpt of an Input object definition:

```
<input name=”Ask for car manufacturer”>
  ...
  <sequence usage=”postprocessing”>
    <log link=”#Car manufacturer”/>
  </sequence>
</input>
```

Preconditions

Preconditions are defined using `<expression usage=”precondition”>`, `<variable usage=”precondition”>`, `<collection usage=”precondition”>`, or `<script usage=”precondition”>` children.

As an example of using an Expression object as a precondition, see the following excerpt of an Input object definition:

```
<input name=”Ask for car model”>
  <expression function="isnull" usage="precondition">
    <argument object="#Car model"/>
  </expression>
  ...
</input>
```

Copyright © 2001-2011 Voxeo Germany GmbH

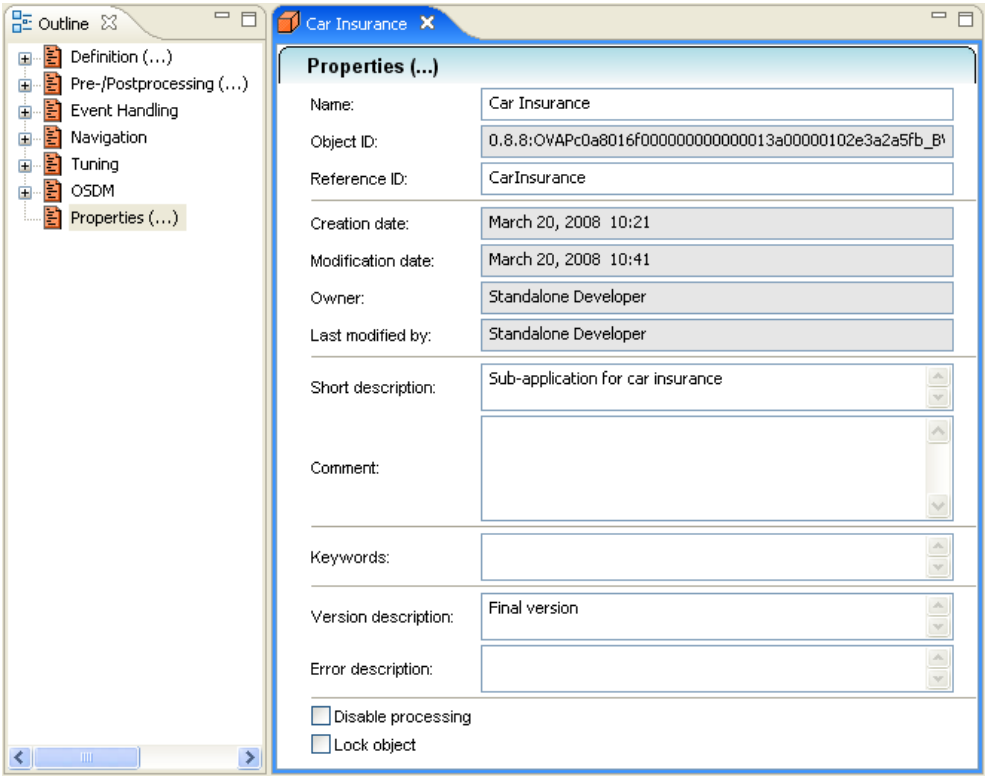
Properties

Overview

Each object is configured by a set of individual properties that are specific for the object type it belongs to. These type-specific properties are defined in the **Definition** section of the corresponding object editor.

Besides these properties all objects have a set of standard properties for the identification and documentation of the object within a project. These standard properties can be viewed and modified in the **Properties** section of the object editors.

Note: In Desktop for Web, you can also edit the standard properties by right-clicking an object in the **Dialog Designer** or in the **Object Browser**, and clicking **Properties** in the respective context menu.



Copyright © 2001-2011 Voxeo Germany GmbH

Properties

The **Properties** section contains the following information:

Property	Description	Size Limit in Bytes
Object ID	This ID is generated by VoiceObjects and cannot be changed. Please note that the actual object ID is only the part between *OVAP* and *_BVO_*. This is important when searching for a specific object ID.	
Reference ID	Unique ID that can be set to reference the object. By default this ID is the same as the object ID. When objects are used as parameters for external programs, plug-ins, or scripts, the reference ID can be used to refer to these objects within the external code. The reference ID can be changed to a custom string that is shorter and easier to understand than the default. The reference ID of an object must be unique within the project.  **Caution:** Only alphanumeric characters (a-z, A-Z, 0-9) and underscores are allowed. No special characters like blanks, '?', '!', '%', '<', '>' etc., and no characters coming from character sets other than ISO-8859-1 can be used. Note that the ID is not case-sensitive, so you cannot have two reference IDs in your project that only differ in case. The reference ID may not be an empty string.	255
Creation date	Time stamp indicating when the object was first created. This is automatically set by VoiceObjects and cannot be changed.	
Modification date	Time stamp indicating when the object was last modified. This is automatically set by VoiceObjects and cannot be changed.	
Owner	Name of the user who created the object. This is automatically set by VoiceObjects and cannot be changed.	
Last modified by	Name of the user who last modified the object. This is automatically set by VoiceObjects and cannot be changed.	

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description	Size Limit in Bytes
Short description	Short description for the object. This information shows up as a tool tip in the **Dialog Designer** and **Object Browser** when pointing to the object. The short description should always be entered, providing a quick overview of the object´s functionality. Using descriptions makes it easier to manage projects.	4000
Comment	A comment associated with this object. Can be used to add a temporary note to an object during the development phase. In the **Dialog Designer** this is indicated by a comment icon along with a tool tip containing the comment, in case comments are activated in the **Display Options**.	8000
Keywords	Keywords associated with this object. Can be used to find objects more easily in the **Object Search**.	4000
Version description	Version-specific information for this object. May be used to document changes made to an object from one version to another.	8000
Error description	Error description for this object. If the object definition is invalid or changes were made to the object definition because of errors in the application, these changes can be documented in this field. This allows the developer to track the history of issue-related changes.	4000
Disable processing	Disables the processing of the object. This is particularly useful during the development and test cycles of a project. Note that disabling the processing of an object affects every location where it is used in a dialog flow.	
Lock object	Locks the object, so that users other than the owner of the lock cannot make any changes to it. Users with higher roles may break such locks. For information on how this interacts with locking whole projects or project versions see Chapter 3 – *Repository Browser* in the *Desktop for Eclipse Guide* and Chapter 2 – *Working with Projects* in the *Desktop for Web Guide*, respectively.	

Note: In case of the fields **Short description**, **Comment**, **Keywords**, **Version description**, and **Error description** using UTF-8 characters is allowed, but may reduce the given size limit, as some characters use more than one byte.

Copyright © 2001-2011 Voxeo Germany GmbH

Note: When using MaxDB as the Metadata Repository DBMS the following limits apply to the size of the standard properties described above:

Reference ID: 255 bytes; **Short description**: 500 bytes; **Comment**: 500 bytes; **Keywords**: 255 bytes; **Version description**: 500 bytes; **Error description**: 255 bytes.

Copyright © 2001-2011 Voxeo Germany GmbH

Recording

Overview

The Recording object is a special kind of input that collects an audio or video message from the caller. The caller is prompted for a message which is recorded with the Recording object. After a successful recording session, the recorded audio or video stream can be stored as a single file on a file server. If required, recordings stored on the server can afterwards be written to a database using a Connector object.

The Recording object provides multiple termination options which apply either automatically or on demand by the caller. A recording session ends when one of the following conditions is met: the specified interval of final silence occurs (only for audio recording), the maximum recording duration is reached, a DTMF key is pressed by the caller (if DTMF recognition is activated), or the caller hangs up.

Due to its nature the Recording object is only relevant for the voice and video channel; in the text and Web channel it will not be processed. For further information on channels refer to Chapter 10 – *How to Support Multiple Phone Channels* in the *Design Guide*.

The Object Definition below covers the configuration of the Recording object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Recording object belongs to the object category Actions.

Dialog Flow Scenario

The first dialog flow example demonstrates a video recording session within a video mail service. The caller is informed that the called party is not available and is asked to leave a message. The message will be recorded and the recorded video file is stored in a message store.

Object – Caller	Dialog Flow
	You have reached the video mail of Tim Miller. *I'm currently not available to take your call.* *Please leave a message after the beep.* [Beep (generated by media platform)]
Caller	*Hi Tim! This is Jeff, I would like to tell you …* *Bye!*
	< Recorded message is stored as a video file >

The next dialog flow illustrates the use of a Recording object as part of a personal voice calendar service. It describes how a caller adds a new appointment to the personal calendar by using two subsequent Recording objects to record the subject and details of the appointment. Instead of a standard beep tone a special sound is used before the recording starts. In the continuing dialog flow the recorded audio files are replayed to the caller for verification as well as a final confirmation including the date and time, which is recognized with an Input object using multi-slot recognition.

Copyright © 2001-2011 Voxeo Germany GmbH

Object – Caller		Dialog Flow
		< Connecting to the database system to load the caller profile and the personal calendar based on the supplied caller's number (ANI) >
		*Hello Susan!* *Welcome to your personal voice calendar.*
		*Please say or key in your five-digit PIN.*
 Caller		*7 4 7 0 1*
		*What can I do for you?*
 Caller		*Add an appointment for* *tomorrow morning from nine to ten.*
		*Enter the appointment subject.* [ Special Sound]
 Caller		*Office of Bob Summerfield*
		< Recorded subject file is stored >
		*Now enter the appointment details.* [ Special Sound]
 Caller		*Clarify the legal issues with Bob …*
		< Recorded details file is stored >
		*Do you want to hear a replay?*
 Caller		*Yes*
	Text	*Subject:*
		[Recorded Subject file]
	Text	*Details:*
		[Recorded Details file]
		*Do you want to change anything?*
 Caller		*No, thanks.*
	Text	*The new appointment on*
		*June 30th [Variable]*
	Text	*From*
		*9 A.M. [Variable]*
	Text	*To*

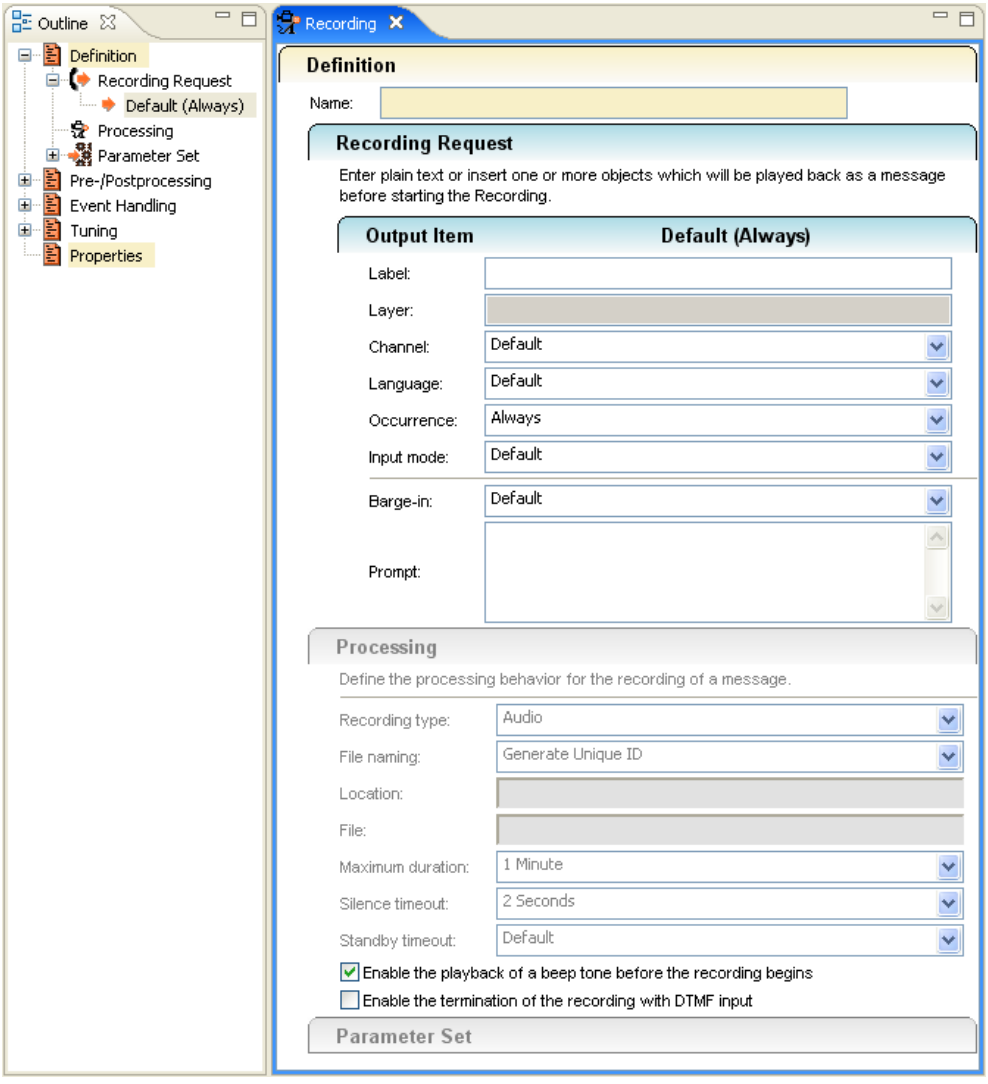
Copyright © 2001-2011 Voxeo Germany GmbH

Object – Caller		Dialog Flow
		10 A.M. [Variable]
	Text	*has been added under the subject*
		[Recorded Subject file]
		Do you want to add another appointment? …

Object Definition

The **Definition** of the Recording object provides the following sections:

- Recording Request
  To specify optional output to prompt the caller to say something.
- Processing
  To configure the recording operation behavior.
- Parameter Set
  To optionally specify additional parameters.

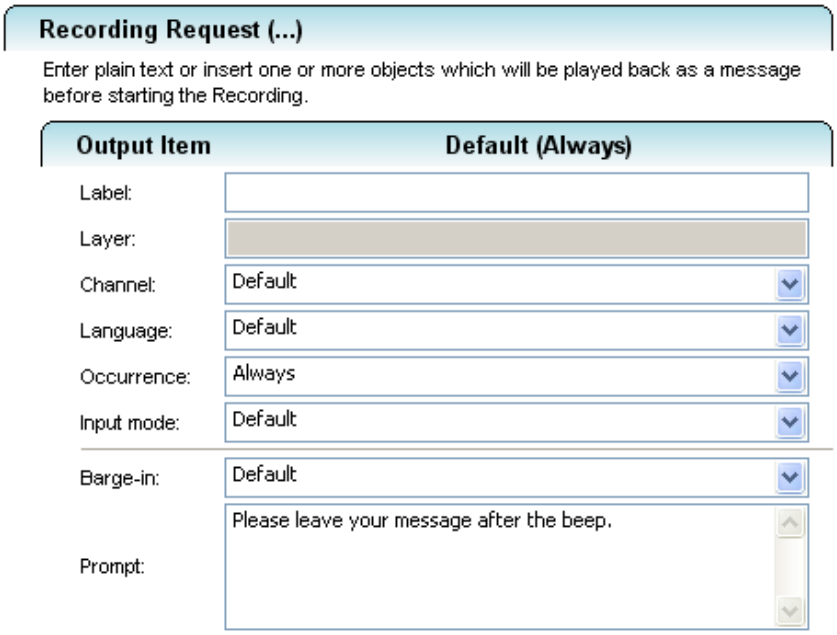


Copyright © 2001-2011 Voxeo Germany GmbH

For information regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Tuning*, and *Properties* in this *Object Reference*.

Recording Request

In the **Recording Request** section, an optional output can be specified that is played to the caller before the recording session starts. It is typically used to prompt the caller to leave a message or to say something.



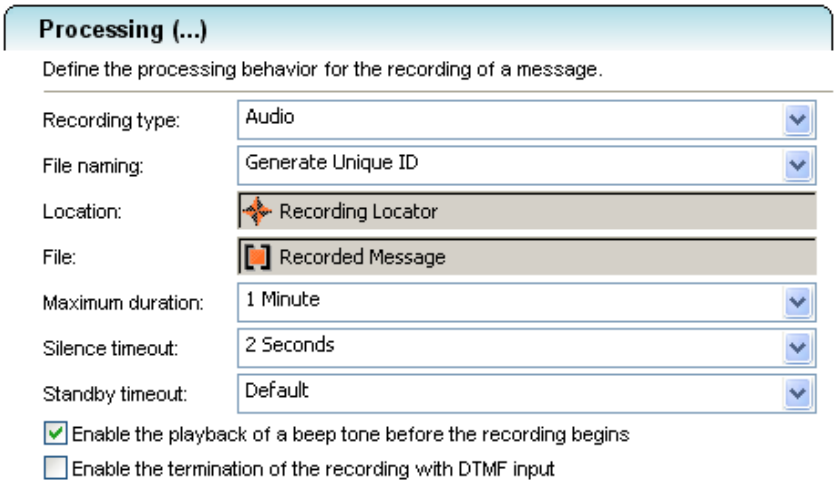
By default, a beep tone is provided by the media platform (see Processing) and should not be included inside the output definition. If a beep tone other than the default one is desired, an Audio or Video object can be included as an alternative at the end of the output stream (see the voice calendar example in Dialog Flow Scenario). In this case, the automated beep tone must be disabled, otherwise both are played back to the caller (see Processing).

For more details on output definitions see the Output object in this *Object Reference*.

Processing

The **Processing** section specifies the processing behavior of the Recording object. It consists of three parts – the first one specifies the handling of the recorded file, the second one is about timing constraints, and the third one about processing options.

Copyright © 2001-2011 Voxeo Germany GmbH



The first part of the **Processing** section provides the following properties to specify the handling of the recorded file:

Property	Description
Recording type	Offers the two choices *Audio* and *Video*. Depending on the type of recording needed in your application, select one of the two entries.
File naming	Offers two different ways of supplying a filename for the recording file: *Generate Unique ID:* The server dynamically generates a unique name including an extension for the recording file and provides this name in the Variable object specified in the **File** field (see below). The generated ID is a 40-character string. The filename is constructed as a concatenation of the ID and the file extension (typically *.wav* for audios and *.3gp* for videos) corresponding to the active media platform driver specification. Example: `0VAPac161748000000000000013b0000000f71aa0903e.wav` *Defined by Object:* The filename without an extension (this is set by the current media platform driver) is provided with the value of a Variable, Expression, Script, or Layer object that is specified within the **File** field. In case of audio recordings, the recorded stream is typically transferred to the server and then stored as a file. Video recordings, on the other hand, are typically stored directly on the media platform, without being sent to the server. This depends on the active media platform. In any case, the file is stored in the location specified through the **Location** and **File** settings (see below) of the **Processing** section. ⓘ ▸▸ **Note:** If a file with the given name already exists at the specified location, it will be overwritten. It is crucial that the storage of recorded audio or video files works in concurrent call sessions, which requires that the user-defined generation of filenames (defined by object) combined with the corresponding file location must be unique for each call session.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Location	Specifies a Resource Locator object that defines the file server and the corresponding directory into which the recorded audio or video stream is stored (**Physical path** field in the Resource Locator object definition), as well as the URL by which those recorded files can be accessed at call time for play back (**URL** field). The latter can, for example, be used to offer the caller a replay of the recorded message and if desired a new recording session which replaces the previous message (see second Dialog Flow Scenario). It is possible to use ftp:// or http:// URIs in the **Physical path**, in which case the recording is uploaded to the specified FTP location or submitted to the specified address, which would typically identify a CGI script that stores the recording somewhere. **Note:** Some media platforms can store the recorded files locally, without transferring them to VoiceObjects Server, which will improve the processing performance of the Recording object. For more information on which platforms do this, refer to Appendix A – *Media Platform Drivers* in the *Deployment Guide*.
File	Specifies the name of the audio or video file to be stored and is supplied with the value of an Expression, Layer, Script, or Variable object. It can also serve as a container in which VoiceObjects Server stores the unique filename it generated (see above). In this case only a Variable object can be used to store the filename. The audio and video file formats in which the recording is stored depend on the media platform used. For most of the platforms supported by VoiceObjects, it is a wav file with the file extension *.wav* for audios, and a 3GP file with the file extension *.3gp* for videos.

To play back a recorded video file, create a Video object and link the Variable object specified in the **File** field of the Recording object into the **File** field of the Video object, setting **Extension** to *None*. Then embed the Video object in an Output object and insert it into the dialog flow.
Recorded audio files can be played back directly by embedding the Variable object in an Output object and using the **Formatting** to play the content back as an audio file (through the text-to-audio feature of the Format object). See the Audio and Format object for more information on this.

The second part of the **Processing** section provides the following properties to specify timing constraints:

Property	Description
Maximum duration	Limits the maximum length of the recording session. The interval is a time designation and is defined to begin immediately after the playback of the **Recording Request** content (including the generated beep tone if enabled). By default, the value for the maximum duration is set to *1 Minute*. When using a Variable, Expression, Script, or Layer object to define the duration interval

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
	at call time, or when typing in a custom value instead of selecting one of the values from the drop-down list, the value needs to be a positive number and will be applied as seconds, e.g. for a custom duration of 5.5 minutes, type in 330. **Tip:** If the caller speaks longer than the time interval defined in the **Maximum duration** field, the event *ASR – Max Processing Time* is activated in case a corresponding event handler is defined. A typical use case would be an event handler that first tells the caller not to exceed the maximum recording duration (by linking an Output object in the handler) and then restarts the recording (by setting **Continuation** to *Return*). **Note:** As mentioned above, a recording session begins immediately after the playback of the recording request including the optional beep tone. As a processing optimization, some media platforms may begin the recording session of an audio message when the caller starts speaking. This is dependent on the individual media platform (more details can be found in the specific documentation of the platform).
Silence timeout	Specifies the time interval of silence required to trigger the termination of the recording session. It indicates to the media platform that the caller has stopped speaking, which will terminate the recording session. Note that this will only work for audio recordings and is again depending on media platform support. The default value of the silence timeout interval is *2 Seconds*. When using a Variable, Expression, Script, or Layer object to define the silence timeout interval dynamically at call time, or when typing in a custom value instead of selecting one of the values from the drop-down list, the corresponding value needs to be a positive number and will be applied as seconds, e.g. for a custom timeout of 5.5 minutes, type in 330. **Note:** When a caller hangs up the phone while the Recording object is being processed by the media platform, the recording session terminates automatically. However, the audio or video stream recorded until hangup is available and will be stored at the specified file location as usual, despite the call disconnection.
Standby timeout	Specifies the maximum time for which the dialog session is kept alive during the recording process. By default this timeout is set to *Default*, which indicates that the value is derived from the service. For more details, refer to Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*. When using an Expression, Layer, Script, or Variable object to define the standby timeout dynamically at call time, or when typing in a custom value instead of selecting one of the values from the drop-down list, the corresponding value needs to be a positive number and will be applied as seconds, e.g. for a custom timeout of 5.5 minutes, type in 330. **Note:** If a caller records for a period longer than the standby timeout, a return to the session is not possible. Thus

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
	designers need to make sure that the standby timeout is at least as long as the maximum duration.

The last part of the **Processing** section specifies options related to the recording:

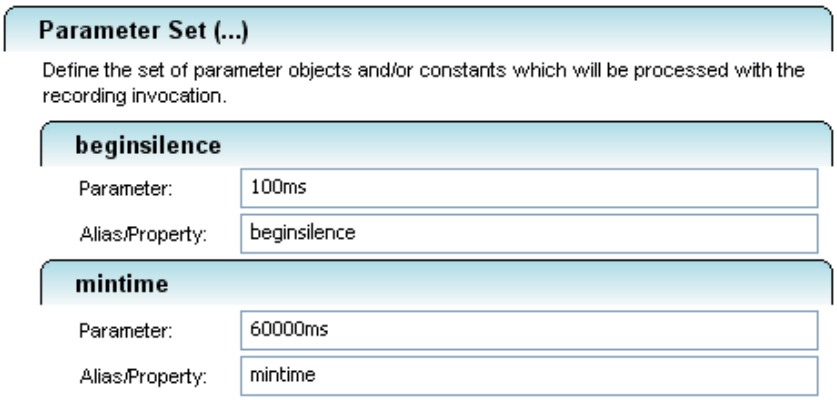
Enable the playback of a beep tone before the recording begins allows you to enable or disable the playing of a beep tone generated by the media platform before starting to record any input (typically only available for audio recordings). By default the check box is selected which means that the beep tone is played. Support for this depends on the active media platform. If a platform does not support the playback of a beep, the setting is typically ignored.

Enable the termination of the recording with DTMF input specifies whether the caller can terminate the recording session by pressing any DTMF key on the phone. You cannot define a specific DTMF key to end the recording. By default the check box is clear, which means that the termination by means of a DTMF key is not available to the caller.

Note: It is also possible to link Expression, Layer, Script, or Variable objects to define the beep tone and termination behavior dynamically at call time. In this case the values of the linked objects need to evaluate to *true* or *false*.

Parameter Set

The **Parameter Set** section defines an optional list of parameters that are passed to the media platform. These parameters allow designers to make use of special extensions to the recording functionality that are provided by a number of platforms.



The example shown here applies to the VoiceGenie platform and indicates that there may be an initial silence of 100ms before the caller needs to speak, and that the entire recording needs to be at least one minute long.

The parameters defined in the **Parameter Set** section are placed into the markup as attributes of the `<recording>` tag, so the VoiceXML code rendered by the server for the example above looks like this:

```
<record beginsilence=”100ms” mintime=”60000ms”> … </record>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Meta Information on Recording Result

After a recording has been made, the media platform provides additional information on the recording result. This information is being made available using the LASTRESULT function through an Expression object, after the Recording object has been processed.

Function	Description
LASTRESULT (duration)	Returns the duration of a recording in milliseconds.
LASTRESULT (size)	Returns the size of a recording in bytes.
LASTRESULT (utterance)	Returns the DTMF key pressed, if the option **Enable the termination of the recording with DTMF input** is selected and the caller presses a DTMF key to terminate the recording.
LASTRESULT (maxtime)	If the recording exceeds the time limit specified in the **Maximum duration** field, this function returns the value *true*, otherwise *false*. Note that this is also realized as an event that can be handled with the *ASR – Max Processing Time* handler without leaving the Recording object.

Usage Examples

Several usage examples of the Recording object are listed below:

- Voice mail or video mail service
- Video or audio dating application with recordings for personal ads
- VoiceGraffitiWall to post messages that other callers may listen to or watch
- Voice calendar service using the Recording object for new appointments made over the phone
- Generic VoicePad module to record and manage personal notes
- Voice banking service where the caller can create transaction templates and save them with a recorded name and description. The template can then be reused to transfer money in ongoing call sessions.
- Voice Forums and transcription applications

VoiceObjectsXML Definition

The Recording object is represented by the VoiceObjectsXML element `<recording>`. It has nine attributes and seven groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

Recording

Attributes

- **`fileNaming`**
  Specifies whether the file name for the recording is to be generated (*generate*), or whether it is provided by an object reference (*object*). If not specified, defaults to *generate*.
- **`location`**
  Defines the location where the recorded audio file is to be stored. Must be a reference to a Resource Locator object.
- **`file`**
  The file name of the audio file. Must be a reference to an Expression, Layer, Script, or Variable object. If `fileNaming` is *generate*, it must be a variable and the generated file name is written into this variable.
- **`recordingType`**
  The type of recording required. Either *audio* or *video*. If not specified, defaults to *audio*.
- **`maxDuration`**
  Defines the maximum length of the recording. A numerical value interpreted as seconds. If not specified, defaults to 60 (one minute).
  May be static text or a reference to an Expression, Script or Variable object.
  For details see Maximum Duration above.
- **`silenceTimeout`**
  Defines the length of silence after which the recording is considered to be complete. An Integer value between 1 and 10, interpreted as seconds.
  May be static text or a reference to an Expression, Script or Variable object.
- **`standbyTimeout`**
  Defines the standby timeout for the Recording object. Either *default* or a numerical value that is interpreted as seconds. If not specified, defaults to *default*.
  May be static text or a reference to an Expression, Script or Variable object.
- **`playBeep`**
  Defines whether a beep tone is played before the recording starts. Either *true* or *false*. If not specified, defaults to *true*.
- **`dtmfTermination`**
  Defines whether the caller may terminate the recording by pressing any DTMF key. Either *true* or *false*. If not specified, defaults to *false*.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Recording object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Recording object.
- **`<output>`**
  Defines the recording request.
- **`<parameterSet>`**
  Defines the parameter set for the Recording object.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<eventHandling>**
  Defines the event handling for the Recording object.
- **<tuning>**
  Defines the tuning settings for the Recording object.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the Recording object.

Example

```
<recording dtmfTermination="true" location="#Recording locator"
file="#Recording">
  <output>
    <outputItem>
      <text>Please record your message after the tone.</text>
    </outputItem>
  </output>
</recording>
```

Object Interoperability

The table below lists all object types that can reference a Recording object:

Icon	Object Name	Use Case Example
	Sequence	A Recording object can be referenced within a Sequence object.
	Menu	A Recording object can be used within the Menu object – one menu choice offering a recording session.
	If	A Recording object can be either in the THEN item or the ELSE item of an If object.
	Case	A Recording object can be used in a WHEN item or the ELSE item of a Case object.
	Goto	A Recording object can be referenced within a Goto object.
	Hyperlink	A Recording object can be the destination of a Hyperlink object, which can be activated on demand by the caller through a specific voice or DTMF command.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of the Recording object should describe what the caller is expected to do, as well as the type of recording (Audio or Video). The name may contain additional hints on the maximum recording length or other options like automatic beep tone generation or enabled DTMF termination. The short description should provide further

Copyright © 2001-2011 Voxeo Germany GmbH

information on how the recorded message is handled and processed, for example any conventions for the filename or the file location. The table below lists a few examples:

Name	Description
Leave a Video Message (Beep)	Caller can leave a video mail message after the beep tone.
Enter Subject [30s]	Subject of the appointment with a maximum length of 30 seconds. The audio file is stored with the following filename convention: [<AppointmentID>_S.wav]
Enter Details [5min]	Details of the appointment with a maximum length of 5 minutes. The audio file is stored with the following filename convention: [<AppointmentID>_D.wav]
Record Dating Ad (Jingle)	Caller can record a personal dating ad after a short jingle is played back. The recording can be terminated with any DTMF keystroke.

Copyright © 2001-2011 Voxeo Germany GmbH

Resource Locator

Overview

The Resource Locator object represents a general access path to any kind of resource in the same way as a URL (Uniform Resource Locator) in the Web environment. It is used to get access to certain file-based resources such as audio, video, grammar, or script files. It defines a physical file location and the corresponding URL for this location. Various objects, including e.g. Audio, Video, Grammar, and Plug-In, use Resource Locator objects to access certain files.

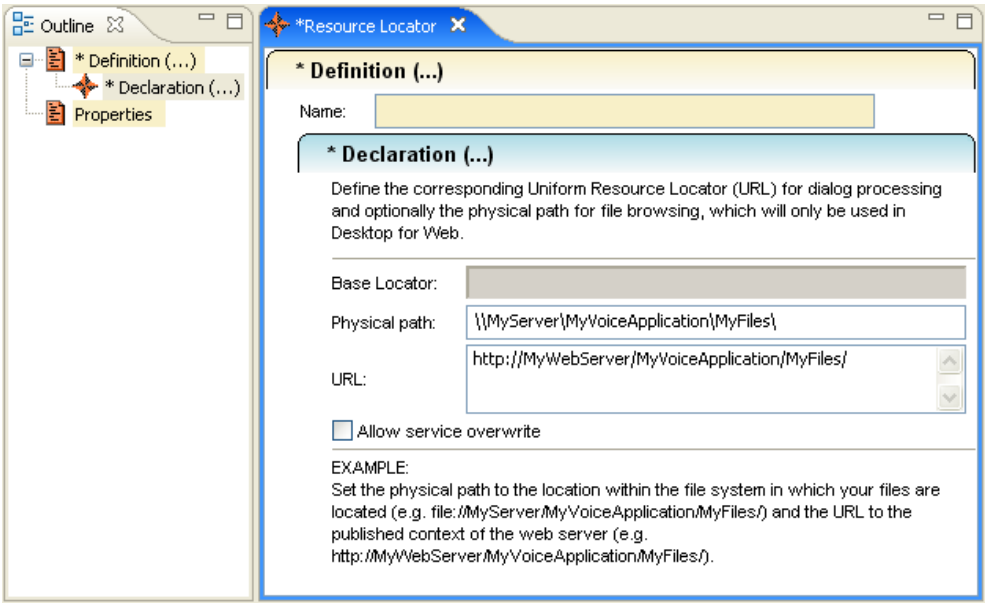
The Object Definition below covers the configuration of the Resource Locator object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Resource Locator object belongs to the object category Resources.

Object Definition

The **Definition** of the Resource Locator object provides one section:

- Declaration
  To denote the physical path for browsing and the corresponding URL (Uniform Resource Locator) for dialog processing.



For information regarding the **Properties** section refer to *Properties* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Declaration

The **Declaration** section denotes the physical path to the resources, the corresponding URL, and a base locator.

In the **Physical path** field, the location within the file system is specified where the targeted files are located, e.g.:

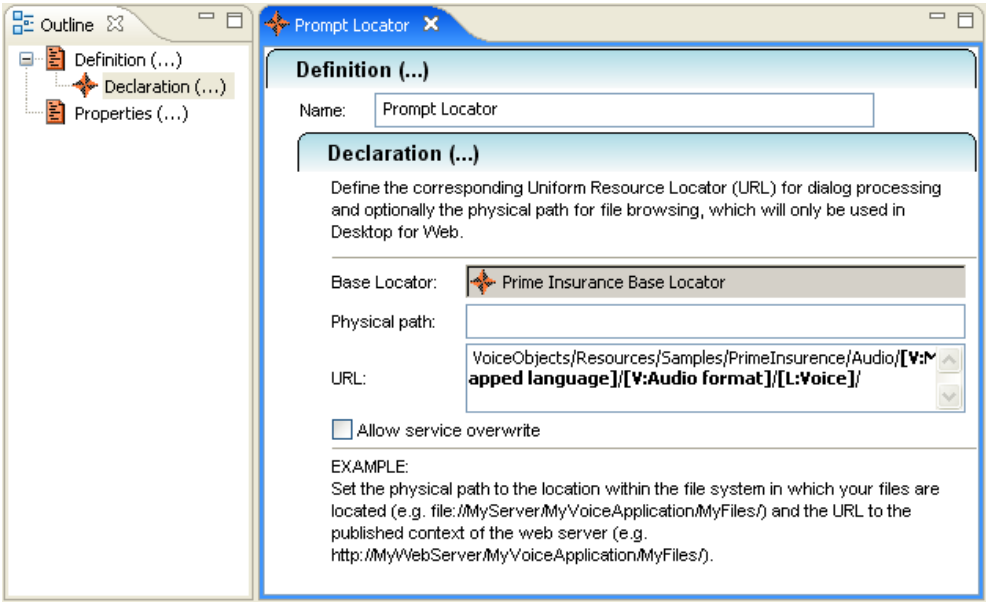
\\MyServer\MyVoiceApplication\MyFiles

The definition of the physical path is optional; this information is mainly used if the user wants to browse for files in VoiceObjects Desktop at development time. In the Recording object, the physical path defines the location of the recorded audio or video files. It can thus be dynamically specified at call time by using a Variable, Expression, Script or Layer object instead of a constant declaration.

In the **URL** field, the published context of the corresponding Web server is specified, e.g.:

http://MyWebServer/MyVoiceApplication/MyFiles/

The definition of the **URL** is required because it is included in the dynamically generated markup documents (e.g. into VoiceXML documents) by the server at call time. The URL can be defined as a combination of text, Variable, Expression, Script, and Layer objects. Variable, Expression, Script and Layer objects are displayed in square brackets with prefix “V:”, “E:”, “X:”, and “L:” respectively.



Object references are typically used within the **URL** for multi-lingual or multi-persona applications, where the different audio files are stored in separate locations on a resource server. By switching the content or state of the corresponding Variable, Expression, Script, or Layer object, the correct audio file is played at call time.

For more information on using dynamic resource locators of this kind see Chapter 7 – *How To Use Layers* in the *Design Guide*.

Caution: Be careful when using Script objects within resource locators. As the script code needs to be evaluated every time the Resource Locator object is used, this may have a severely negative impact on performance.

Copyright © 2001-2011 Voxeo Germany GmbH

Note: Depending on the operating system, the physical path and the URL definition may not allow blanks or other special characters.

In the **Base Locator** field, a parent Resource Locator object may optionally be specified. Both the physical path and the URL defined in this parent Resource Locator object are prepended to the respective entries in the current Resource Locator object. Base Resource Locator objects can be stacked in any number of levels, thus providing a convenient mechanism to operate relative to certain base paths.

Caution: Do not use a Resource Locator object as its own base locator. This will lead to infinite loops and prevent your application from being deployed to VoiceObjects Server.

When the check box **Allow service overwrite** is selected, the definition made in the corresponding Service object overwrites the local definitions of the resource locator. This is useful in particular for base locators, as their definition might change when migrating from a prototype to a production deployment. Defining it in the Service object allows you to make the change independent of the project itself. For more information on the Service definition see Chapter 2 – *Configuring Servers and Services* in the *Deployment Guide*.

Accessing Hosted Resources

When working with Voxeo Evolution you have the option of uploading resources such as audio and grammar files to Voxeo web hosting. Visit the *Files, Logs, Reports* section of your Evolution account for more details.

Once you have uploaded a file, it is accessible through a URL of the format *http://webhosting.voxeo.net/{user ID}/www/...* The exact URL is shown for each file:



To access hosted files within your application it is recommended to create a **Base Locator** that models this URL.

VoiceObjectsXML Definition

The Resource Locator object is represented by the VoiceObjectsXML element `<resourceLocator>`. It has three attributes and one group of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

Resource Locator

Attributes

- **`base`**
  Defines the base locator. Must be a reference to a Resource Locator object.
- **`physicalPath`**
  Text string that defines the physical path used for file browsing (only relevant in VoiceObjects Desktop) or to store a recorded audio file. All paths should use forward slashes.
- **`allowServiceOverwrite`**
  Either *true* or *false*. Indicates whether physical path and URL may be overwritten by defaults defined in the Service object. If not specified, defaults to *false*.

Children

- **+(`<variable>`, `<expression>`, `<script>`, `<text>`, `<layer>`)**
  Defines the URL from which to fetch the resource.

Example

```
<resourceLocator physicalPath="C:" allowServiceOverwrite="true">
  <text> http://127.0.0.1:7077/</text>
  <variable link="#Basepath"/>
  <text>/</text>
  <variable link="#System:Language"/>
  <text>/</text>
</resourceLocator>
```

Object Interoperability

The following table contains all object types that can reference a Resource Locator object:

Icon	Object Name	Use Case Example
	Audio	A Resource Locator object can be referenced within an Audio object.
	Video	A Resource Locator object can be referenced within an Video object.
	Format	A Resource Locator object can be referenced within a Format object.
	Grammar	A Resource Locator object can be referenced within each Grammar item in a Grammar object.
	Connector	A Resource Locator object can be referenced within each connector item in a Connector object.
	Script	A Resource Locator object can be referenced within a Script object.

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Object Name	Use Case Example
	Plug-In	A Resource Locator object can be referenced within a Plug-In object.
	Resource Locator	A Resource Locator object can be referenced within another Resource Locator object as a parent locator.
	Recording	A Resource Locator object can be referenced within a Recording object.
	OSDM	A Resource Locator object can be referenced within the parameter set of any OSDM object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Resource Locator object should indicate which types of resources it is used for. The table below lists three examples:

Name	Description
 Base Locator	Base Resource Locator object providing the initial path (parent) for all other Resource Locator objects.
 Grammar Locator	Resource Locator object for grammar files.
 Claim recording location	Resource Locator object indicating the location of the recorded claim descriptions.

Copyright © 2001-2011 Voxeo Germany GmbH

Script

Overview

The Script object allows the specification of server-side scripting language code to control any kind of business logic within the dialog flow. The script code can either be embedded within the object definition itself, or it can be referenced as an external file source. With the file referencing option an existing script file (e.g. from an operating Web service) can be reused for building a voice service.
VoiceObjects Server comes with a built-in JavaScript 1.5 engine, but any other script engines can be integrated through the so-called *Script bus*. See Appendix C – *How to Use the Script Bus* in the Administration Guide for more information on this.



Caution: Processing script code is considerably more time-consuming than processing Expression objects. Note that the execution will be done within the VoiceObjects Server framework and not on the media platform (e.g. by embedding the script code into the rendered VoiceXML). If a Script object is only to perform simple calculations use an Expression object instead which offers a variety of frequently used functions. Also avoid using Script objects as part of Resource Locator or Layer objects as these tend to be evaluated in each dialog step. Under high call volume, this can impose high memory and CPU load upon VoiceObjects Server.

Any number of parameters from the dialog context can be passed into the Script object. These parameters can be a mix of constants, Variable, Expression, Collection, Layer, or other Script objects and will be transferred with the script invocation. The value of each transferred Variable object will be assigned back to the dialog context when the script execution is finished. During execution, the Script object has its own context. Any assignment to a script variable has no impact on the dialog context before ending the script evaluation.

The Script object has a return value that is defined by the return value of the script instruction last processed. This can be, for example, the return value of a function which is called from the main scripting block, or it can be a script variable which is assigned a certain value. A Script object can therefore be used in the same way as a Variable, Expression, or Layer object within almost any other object to control certain object properties or processing conditions dynamically at call time. When a Script object does not return a value it is handled as a variable with an undefined value (empty string), which is equivalent to NULL. A Script object not returning any defined value can be tested with the Expression function ISNULL (), which will return *true*, or ISNOTNULL (), which will return *false*. In a Boolean context the Script object evaluates to *true* unless its return value is an empty string, 0 (zero), or FALSE (in any combination of upper and lower case letters).

For more information on JavaScript, refer to the JavaScript Language Resources available at http://www.mozilla.org/js/language/.

The Object Definition below covers the configuration of the Script object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Script object belongs to the object category Resources.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Flow Scenario

The following dialog flow example demonstrates the use of a Script object to format a variable for text-to-audio (TTA) output (see the Format object for more information on text-to-audio). The script builds a TTA–XML string that represents a sequence of audio files to read out amounts. Example: a variable content *4901* is transformed into the sequence *4000.wav 900.wav 1.wav*, resulting in the output "*four thousand nine hundred one*".

Object – Caller	Dialog Flow
	*How much money do you want to transfer?*
 Caller	*4901 euros.*
	[Transforms the string "4901" to a TTA – XML structure]
	*You said four thousand nine hundred one. Is that correct?*

The JavaScript sample code for the transformation may look like the following:

```
var result = "<root>";
var tens;
var hundreds;
var thousands;

tens = amount % 100;
hundreds = (amount - tens) % 1000;
thousands = amount - hundreds - tens;

if (thousands > 0)
{
  result += "<row><col name=\"file\">";
  result += thousands.toFixed(0);
  result += ".wav</col></row>";
}

if (hundreds > 0)
{
  result += "<row><col name=\"file\">";
  result += hundreds.toFixed(0);

  if (tens > 0)
  {
    result += ".wav</col></row>";
  }
  else
  {
```

Copyright © 2001-2011 Voxeo Germany GmbH

```
    result += ".wav</col></row>";
  }
}

if (tens > 0)
{
  result += "<row><col name=\"file\">";
  result += tens.toFixed(0);
  result += ".wav</col></row>";
}

result += "</root>";

result;
```

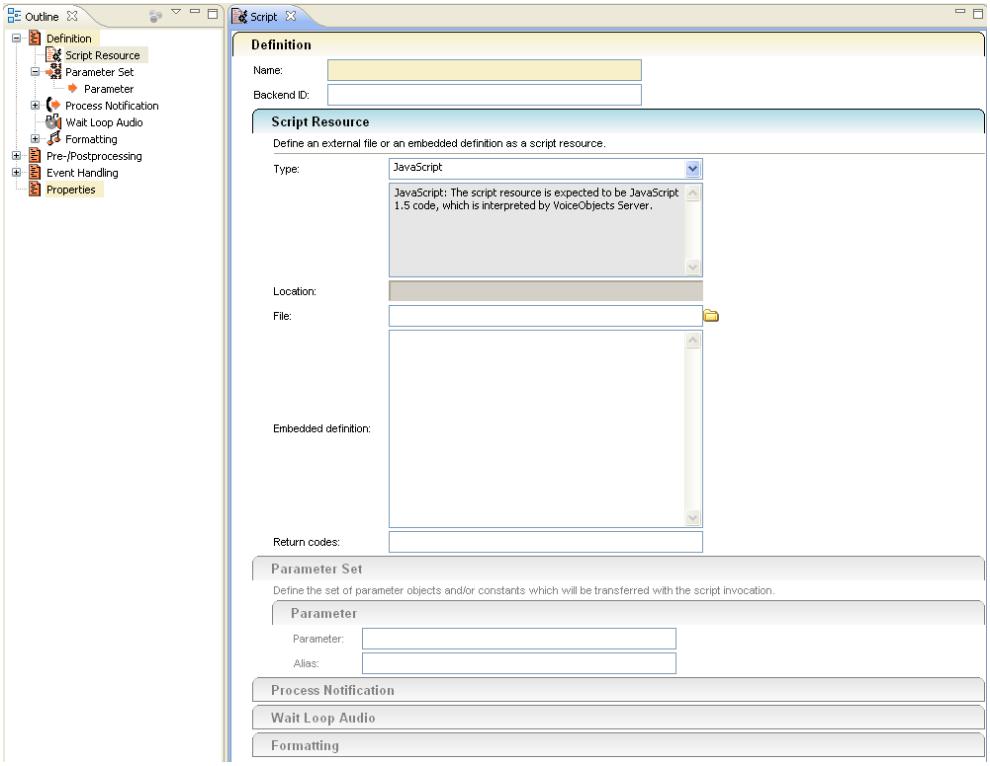
In this example, the variable to pass into the script would be *amount*.

Object Definition

The **Definition** of the Script object provides the following sections:

- Script Resource
  To specify the script code to be processed.
- Parameter Set
  To specify the set of parameter objects and/or constants.
- Process Notification
  To define a message to be played before processing the script.
- Wait Loop Audio
  To define an Audio object to play in a loop while processing the script.
- Formatting
  To define formatting instructions for playback within an Output object.

Copyright © 2001-2011 Voxeo Germany GmbH



The optional **Backend ID** at the top of the **Definition** section can be used to identify the backend component that is accessed by the Script object.

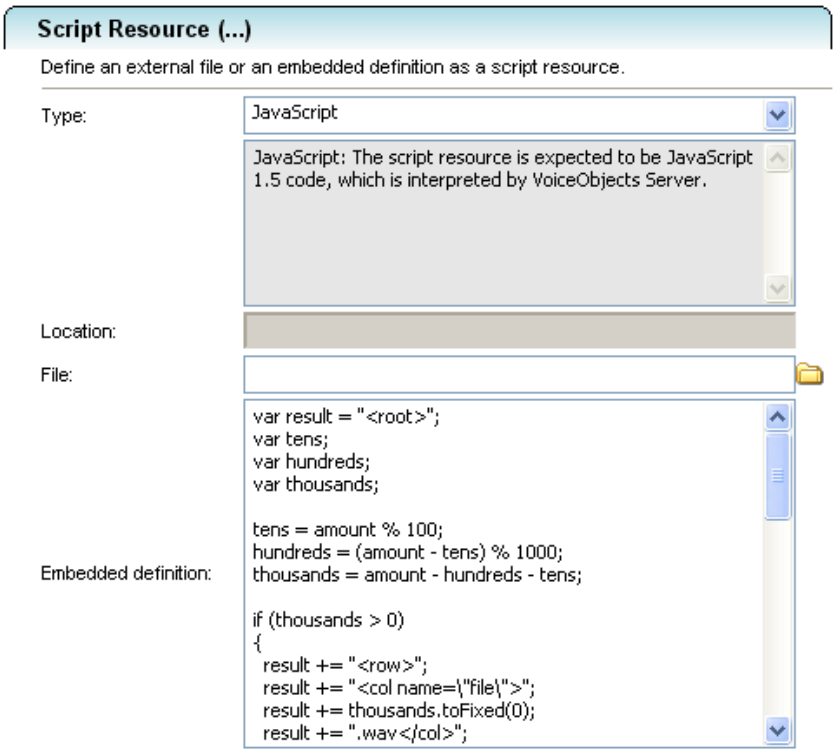
For information regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, or *Properties* in this *Object Reference*.

Script Resource

The **Script Resource** section defines the code to be processed within the Script object. It can either be provided as an embedded definition typed into the **Embedded definition** field (as shown in the example below), or as a reference to a file. In the latter case, the name of the file needs to be provided in the **File** field, and an optional Resource Locator object can be specified in the **Location** field.

Note: If both an embedded definition and a file reference are given, the embedded definition takes precedence.

Copyright © 2001-2011 Voxeo Germany GmbH



The JavaScript resource can also be defined dynamically by using a Variable, Expression, Layer, or another Script object.

The **Type** field allows selecting between the script engines defined through the Script bus. This drop-down list initially consist of only one entry, JavaScript 1.5, which is built into VoiceObjects Server.

A Script object can provide a return value, and thus act as a replacement for an Expression object. The return value of the Script object is defined by the value of the last statement that is evaluated within the JavaScript code. The following two examples demonstrate this.

In the first example, the Script object calls a function defined within the JavaScript code.

```
function addNumbers(x,y) {return x+y}


addNumbers(value1, value2);
```

The two values `value1` and `value2` are passed into the script using the parameter set. The return value of the script then is the sum of these two values. Note that in this example, neither `value1` nor `value2` are modified.

In the second example, an assignment is used as the final statement of the JavaScript code.

```
value1 = value1 + value2;
```

Here the return value of the script is the value of the assignment, namely the sum of `value1` and `value2`. In addition, if `value1` was passed into the script as a variable via the parameter set, then after the script has been processed successfully, this Variable object also contains the sum.

Copyright © 2001-2011 Voxeo Germany GmbH

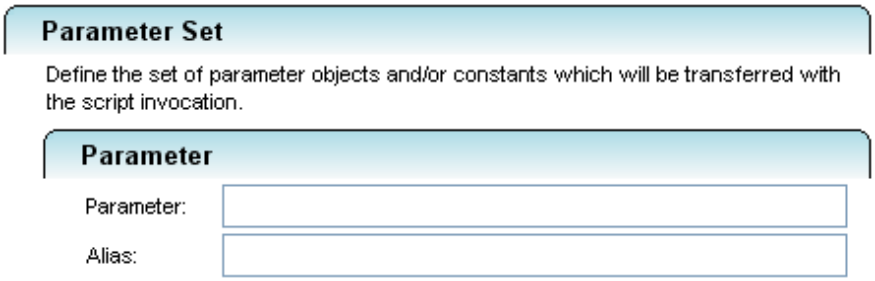
Re-assignments of this type are also possible using a Layer object. In this case, the script needs to return a state ID, so that the layer is switched to the corresponding state.

Should the execution of the JavaScript code fail for any reason (e.g. due to a syntax error in the JavaScript code), an event *Error – Internal* is activated. It can be caught and handled appropriately by using event handling. This can be defined either within the Script object itself, or e.g. in a parent module of the Script object, through inheritance.

You can manually provoke an error by assigning values to the automatically declared variables *errorCode* and *errorMessage* in your code. *errorCode* must be a value other than 0 (zero) to be interpreted as an error, and *errorMessage* can be any string. When the script has been processed and *errorCode* is not 0, the server activates the *Error – Internal* event.

Parameter Set

The **Parameter Set** section defines an optional list of parameters to enable the exchange of processing information with the script code.



Each **Parameter** section provides the following properties:

Property	Description
Parameter	Specifies the value of the parameter. Possible parameter types are Variable, Collection, Expression, Layer, Resource Locator or Script objects, and constants. When using a Resource Locator object as a parameter, its URL definition is passed to the script. The field can be left empty to denote an empty value.
Alias	Optionally specifies the name under which the parameter is accessible within the script code. If left empty, the reference ID of the object defined in the **Parameter** field will be used as the name. This means in particular that for constant values an alias needs to be defined (as they do not have a reference ID). Constant parameters without a defined alias are ignored.

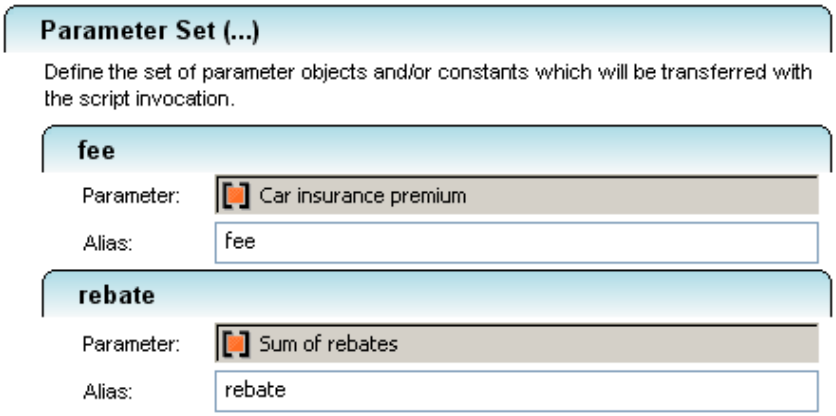
Variables that are passed into the Script object using the parameter set do not need to be declared within the script code. Note, however, that the script engine may interpret the parameters as they are passed to it. This means, for instance, that parameters that “look like numbers” will be interpreted as numbers by the engine even if they were intended as strings. This might for example happen to a value reflecting a date, like *21.02.99*. To prevent the engine from converting this to some number, you can prefix

Copyright © 2001-2011 Voxeo Germany GmbH

the value with a non-numerical character (like “x”) to make it a string, before passing it to the script, i.e. before processing the Script object. Make sure to delete this additional character again within the script code before processing it any further.
In other cases you might need to explicitly cast values back within the script code, e.g. when you pass a variable into the script that currently holds a number, but within the script you want to treat it like a string and perform string operations on it.

After the script code has been processed successfully, all Variable, Collection, and Layer objects are written back. Note that Resource Locator objects can *not* be written back like this.

Any number of parameters can be exchanged with the script code.



Note: JavaScript has a set of reserved words that may not be used as variable names. This implies that these words may also not be used in the **Alias** field. JavaScript reserved words include e.g. *continue*, *debugger*, *default*, *delete*, *false*, *function*, *import*, *package*, *private*, *protected*, *return*, *super*, *switch*, *true*.

In addition to passing parameters, the script code also has access to a certain set of functions. Note that this is only the case for JavaScript code, not for any other script engines attached to the Script bus. The functions are:

Boolean Functions	
Name and Arguments	**Description**
vo.and (arg1, arg2 [, .., arg6])	Returns *true* if all arguments are true; otherwise returns *false*.
vo.between (arg, lowerLimit, upperLimit)	Returns *true* if the specified argument is between the lower and upper limit, or equal to the lower or upper limit; otherwise returns *false*.
vo.in (arg, delimitedString, [separator])	Returns *true* if the specified argument is within the list of entries provided within the delimited string; otherwise returns *false*. The default separator is a comma, but a custom separator can optionally be provided as a third argument.

Copyright © 2001-2011 Voxeo Germany GmbH

Boolean Functions	
Name and Arguments	**Description**
vo.matchesRegExp (txt, regExp, [caseSensitive])	Returns *true* if *txt* matches the regular expression *regExp*, otherwise returns *false*. The optional parameter *caseSensitive* indicates whether the comparison should be case sensitive (*true* or false; default is *false*).
vo.nand (arg1, arg2 [, .., arg6])	Returns *true* unless all its arguments are true; otherwise returns *false*. NAND is the logical complement of AND.
vo.nor (arg1, arg2 [, .., arg6])	Returns *true* if none of its arguments is true; otherwise returns *false*. NOR is the logical complement of OR.
vo.not (arg)	Returns the reversed value of its argument.
vo.notbetween (arg, lowerLimit, upperLimit)	Returns *true* if the specified argument is not between the lower and upper limit; otherwise returns *false*.
vo.notin (arg, delimitedString, [separator])	Returns *true* if the specified argument is not within the list of entries provided within the delimited string; otherwise returns *false*. The default separator is a comma, but a custom separator can optionally be provided as a third argument.
vo.or (arg1, arg2 [, .., arg6])	Returns *true* if any argument is true; otherwise returns *false*.
vo.xor (arg1, arg2 [, .., arg6])	Returns *true* if exactly one argument is true; otherwise returns *false*. XOR (arg1, arg2 [, ..., arg6]) stands for exclusive OR (arg1, arg2 [, ..., arg6]).

Information Functions	
Name and Arguments	**Description**
vo.isAfter (date1, date2)	Returns *true* if *date1* occurs chronologically after *date2*; otherwise returns *false*. The parameters *date1* and *date2* must be specified in the format yyyyMMdd or yyyyMMddhhmmss.

Copyright © 2001-2011 Voxeo Germany GmbH

Information Functions	
Name and Arguments	**Description**
vo.isBefore (date1, date2)	Returns *true* if *date1* occurs chronologically before *date2*; otherwise returns *false*. The parameters *date1* and *date2* must be specified in the format yyyyMMdd or yyyyMMddhhmmss.
vo.isLogical (arg)	Returns *true* if the specified argument is a logical value (*true* or *false*, regardless of capitalization); otherwise returns *false*.
vo.isNotNull (arg)	Returns *true* if the specified argument is defined; otherwise returns *false*.
vo.isNull (arg)	Returns *true* if the specified argument is undefined or empty; otherwise returns *false*.
vo.isNumber (arg)	Returns *true* if the specified argument is a number; otherwise returns *false*.
vo.isText (arg)	Returns *true* if the specified argument is a text; otherwise returns *false*. A number is not interpreted as text, thus ISTEXT(5) returns *false*.
vo.isValidDate (date)	Returns *true* if date is a valid date otherwise returns *false*. The parameter *date* must be specified in the format yyyyMMdd or yyyyMMddhhmmss.

Arithmetic Functions	
Name and Arguments	**Description**
vo.abs (nbr)	Returns the absolute value of the specified number. The absolute value of a number is the number without its sign.
vo.ceil (nbr)	Returns a number rounded up, away from zero (0).
vo.floor (nbr)	Returns a number rounded down, toward zero (0).
vo.ln (nbr)	Returns the natural logarithm of the specified number. Natural logarithms are based on the constant “e” (2.71828182845904). The specified number must be positive.

Copyright © 2001-2011 Voxeo Germany GmbH

Arithmetic Functions	
Name and Arguments	**Description**
vo.pow (nbr, power)	Returns the value of the specified number to the power specified by the second argument.
vo.rand ( )	Returns a Float random number between 0 (inclusive) and 1 (exclusive).
vo.randint (start, end)	Returns a random Integer value between *start* (inclusive) and *end* (exclusive). Both arguments *start* and *end* must be Integer values, and *end* must be greater than *start*.
vo.round (nbr, nbrDigits)	Returns the result of rounding the specified number to at most *nbrDigits* decimal digits.
vo.sign (nbr)	Returns the sign of the specified number. Returns 1 if the number is positive, zero (0) if the number is 0, and -1 if the number is negative.
vo.sqrt (nbr)	Returns the square root of the specified number. The number must be positive or zero (0).
vo.trunc (nbr, nbrDigits)	Returns the result of truncating the specified number to at most *nbrDigits* decimal digits.

Text Functions	
Name and Arguments	**Description**
vo.find (findTxt, txt, startPos)	Returns the index at which *findTxt* was found within *txt*, starting the search at position *startPos* (where the first character is position 1). If *findTxt* could not be found, 0 (zero) is returned.
vo.insert (oldTxt, startPos, newTxt)	Returns a modified version of the string *oldTxt* in which the string *newTxt* has been inserted, starting at position *startPos*.
vo.lower (txt)	Returns a new string in which all uppercase letters of the specified string *txt* are converted to lowercase.

Copyright © 2001-2011 Voxeo Germany GmbH

Text Functions	
Name and Arguments	**Description**
vo.ltrim (txt, [char])	Returns a modified version of the string *txt* in which white space characters (blanks and tabs only) are removed at the left end. Optionally a *char* can be specified which will be removed instead of white space characters.
vo.mid (txt, startPos, nbrChars)	Returns a substring for the specified string *txt*, starting at position *startPos* for the specified number of characters.
vo.replace (oldTxt, startPos, nbrChars, newTxt)	Returns a modified version of the string *oldTxt* in which the substring starting at position *startPos* containing *nbrChars* characters has been replaced by the string *newTxt*. Example: replace(hello, 3, 2, r) = hero
vo.rtrim (txt, [char])	Returns a modified version of the string *txt* in which white space characters (blanks and tabs only) are removed at the right end. Optionally a [*char*] can be specified which will be removed instead of white space characters.
vo.substitute (string, pattern, replacement, [patternIsRegExp])	Returns a string in which all occurrences of the *pattern* are substituted by the *replacement* string. The original *string* is provided as first argument and will not be modified. The optional parameter *patternIsRegExp* indicates whether *pattern* has to be interpreted as a regular expression (*true* or *false*; default is *true*).
vo.trim (txt, [char])	Returns a modified version of the string *txt* in which white space characters (blanks and tabs only) are removed at the left and right end. Optionally a [*char*] can be specified which will be removed at both ends instead of white space characters.
vo.upper (txt)	Returns a new string in which all lowercase letters of the specified string *txt* are converted to uppercase.

Copyright © 2001-2011 Voxeo Germany GmbH

Date & Time Functions	
Name and Arguments	**Description**
vo.addDate (date, increment, [unit])	Adds the *increment* to *date* and returns the new date in the format yyyyMMddhhmmss. The parameter *date* must be specified in the format yyyyMMdd or yyyyMMddhhmmss. If *increment* is less than 0, *date* is decreased accordingly. The optional parameter unit can be used to define the unit as one of "years, months, days, hours, minutes, seconds" (default is days).
vo.convertDate (txt, outputFormat, [inputFormat], [languageCode])	If *txt* is a date following the format yyyyMMdd or yyyyMMddhhmmss this returns the date in the specified *outputFormat* using the Java SimpleDateFormat convention. If *txt* is a date following a custom format the optional parameter *inputFormat* must be used to specify the format syntax in order to enable the correct conversion internally (e.g. if *txt* contains "23102008" the *inputFormat* should be set to "ddMMyyyy"). The optional parameter *languageCode* can be used to define the language code, based on which the regional setting is determined (e.g. *en-US* or *de-DE*; default is the language code of the current call). This information will be used to localize the returned date format if necessary, e.g. if the call is processed in *en-US* a date may contain the abbreviation *AD*, when switching the language code to *de-DE* this would be changed to *n.Chr*.
vo.lastDayInMonth (date)	Returns the last day in the month represented by *date* in the format yyyyMMddhhmmss. The parameter *date* must be specified in the format yyyyMMdd or yyyyMMddhhmmss.
vo.nextWeekDay (date, weekday, [languageCode])	Returns the date of the next weekday, which is represented by an integer from 1 (first day in week) to 7 (last day in week), in the format yyyyMMddhhmmss starting from *date*. The parameter *date* must be specified in the format yyyyMMdd or yyyyMMddhhmmss. Depending on the regional setting this may differ as the week in Europe typically starts with Monday, whereas the US week starts with Sunday. The optional parameter *languageCode* can be used to define the language code based on which the regional setting is determined (e.g. *n-US* or *de-DE*; default is the language code of the current call).

Copyright © 2001-2011 Voxeo Germany GmbH

Date & Time Functions	
Name and Arguments	**Description**

vo.now ([format], [offsetToUTC])	Returns the current timestamp in the specified format using the Java SimpleDateFormat convention. Sample formats and return values are:
	`yyyy.MM.dd G 'at' HH:mm:ss z`
	`2001.07.04 AD at 12:08:56 PDT`
	`EEE, MMM d,''yy`
	`Wed, Jul 4,'01`
	`h:mm a`
	`12:08 PM`
	`hh 'o''clock' a, zzzz`
	`12 o'clock PM, Pacific Daylight Time`
	`K:mm a, z`
	`0:08 PM, PDT`
	`yyyyy.MMMMM.dd GGG hh:mm aaa`
	`02001.July.04 AD 12:08 PM`
	`EEE, d MMM yyyy HH:mm:ss Z`
	`Wed, 4 Jul 2001 12:08:56 -0700`
	`YyMMddHHmmssZ`
	`010704120856-0700`
	If no format is specified, the default format **`yyyyMMddHHmmss`** is used.
	The optional second parameter offsetToUTC can be used to switch to a different time zone when interpreting the format parameter. The offset must be provided in the format [-]HHMM.
	A detailed definition can be found at: *http://java.sun.com/javase/6/docs/api/java/text/SimpleDateFormat.html*

vo.timeBetween (date1, date2, [unit])	Returns the number of days between *date1* and *date2*. The parameters *date1* and *date2* must be specified in the format yyyyMMdd or yyyyMMddhhmmss. The optional parameter *unit* can be used to define the unit as one of "years, months, days, hours, minutes, seconds" (default is days). The calculation ignores everything below *unit* (e.g. if *unit* is set to hours, minutes and seconds of *date1* and d*ate2* will be ignored).
vo.timeZoneInfo ()	Returns a collection with four columns. The first column *Offset* shows the offset in hours to add to UTC to get local time (e.g. "-0500" or "0230"). The second column *StandardName* contains the standard name of the local timezone (e.g. *PST* or *CET*). The third column *DaylightName* shows the daylight name of the local time zone (e.g. *PDST* or *CEDT*). The fourth column *DSTInEffect* indicates whether the local time is Daylight Saving Time or not (*true*, *false* or *unknown*).

Copyright © 2001-2011 Voxeo Germany GmbH

Date & Time Functions	
Name and Arguments	**Description**
vo.timer (option)	Provides a stopwatch functionality. When called with argument *start* it resets the timer. When called with argument *interval* it returns the time difference between now and the last *start* or *interval* request (in milliseconds).
vo.weekDay (date, [languageCode])	Returns the weekday of *date* represented by an integer from 1 (first day in week) to 7 (last day in week). The parameter *date* must be specified in the format yyyyMMdd or yyyyMMddhhmmss. Depending on the regional setting this may differ as the week in Europe typically starts with Monday, whereas the US week starts with Sunday. The optional parameter *languageCode* can be used to define the language based on which the regional setting is determined (e.g. *en-US* or *de-DE*; default is the language code of the current call).

Statistics Functions	
Name and Arguments	**Description**
vo.eventCounter (event, [scope])	Returns the number of times the specified event has occurred during the most recent input state (*scope* = *local*) or during the entire dialog (*scope* = *global*). If the scope is not defined, it defaults to *global*. The third argument allows distinguishing between the *collection* and *confirmation* phase of OSDM objects. The default value is *global*, covering both phases. Example: *eventCounter(nomatch)* will return the number of No Match events that have been triggered in the current dialog. For more information on the different events, refer to Event Handling in this *Object Reference*.
vo.eventList (scope)	Returns a comma-separated list of events, which have occurred either in the most recent input state (*scope* = *local*) or in the entire dialog (*scope* = *global*). The second argument allows distinguishing between the *collection* and *confirmation* phase of OSDM objects. The default value is *global*, covering both phases.

Copyright © 2001-2011 Voxeo Germany GmbH

Statistics Functions	
Name and Arguments	**Description**
vo.lastResult (type, [index], [phase])	Returns information on the most recent recognition result, as well as information on results from recordings and transfers. The first argument can be *confidence*, *interpretation*, *length*, recordingpath, or recordingurl (for recognition results)*, inputmode* (for recognition results or transfers), *utterance* (for recognition results, transfers, and recordings), *markname* or *marktime* (for prompt markers and barge-in point detection), *size* or *maxtime* (for recordings), *duration* (for recordings and transfers), *transferstatus* (for transfers), and *validation* (for recognition results with server-side input validation, using the regex: … :regex syntax for TTG grammars in the text and web channels). For recognition results, if the recognizer returns more than one result, the optional second argument can be used to specify the desired result. *Index* must then be a number between 1 and *Max N-Best* (a tuning property). If *index* is greater than the maximum number of results, the function returns an empty string. The type *length* returns the number of results. The third argument can be used to distinguish between the *collection* (the default) and *confirmation* phase in OSDM objects. If it is set, the second argument must also be defined. *Markname* returns the name of the last prompt marker processed (a `<mark>` statement in an Output item). *Marktime* returns the time elapsed since the last marker was processed. For more information on the recording and transfer related types, see the Recording and Transfer object documentation. *Recordingpath* returns the path and *recordingurl* returns the URL of the last recorded utterance, in case utterance recording was enabled. *Validation* returns a comma-separated list of all slot names for which input validation (against the regular expression for that slot) failed. When called with a slot name as the second argument, the return value is *true* if validation was successful, *false* if it failed.
vo.lastTransition (value)	Returns the mode (for *value* = *mode*) or type (for *value* = *type*) of the most recent dialog transition. Possible modes are *Standard Navigation*, *Auto Advance*, *Recognition*, *Hyperlink*, *Hangup*, and *Error*. The type holds more fine-grained information on how the caller left the last input state.

Copyright © 2001-2011 Voxeo Germany GmbH

Business Task Functions	
Name and Arguments	**Description**
vo.startTask (task)	Starts the business task that is specified in the first argument.
vo.finishTask (task, status)	Finishes the business task that is specified in the first argument with the *status* specified in the second argument. This can be one of *backEndError*, *businessLogic*, *callerAbort*, *recognitionFailure*, *technicalError*, or *complete*. Optionally, the first argument can be *all* to finish all tasks or *active* to finish the currently active task. All values are case-sensitive.
vo.taskRefID ()	Returns the reference ID of the currently active business task. If no task is active an empty string is returned.
vo.taskStatus (task)	Returns the most recent status of the business task specified in the first argument. If the task has not been finished yet, it returns *active* or *inactive*. If the task has never been started, it returns *notStarted*.
vo.taskLast (mode)	Returns information about the last business task processed within the current dialog session. Valid modes are: • **completion** Completion result of the business task; either *complete* or *incomplete* • **duration** Duration of the business task in milliseconds • **occurrence** Occurrence of the same business task within this dialog session • **refID** Reference ID of the business task • **status** Status in which the business task was finished • **steps** Number of input steps processed while the business task was active. This information is only available if System DB logging is active for the service. If no business task has been finished so far, the empty string is returned for all modes.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions	
Name and Arguments	**Description**
vo.bargein ([arg])	When used without an argument, indicates whether barge-in is activated by default (*true*) or not (*false*). When used with an argument (either *true* or *false*), it sets the default barge-in behavior.
vo.applyConfiguration (configurationXML)	Applies the assignments defined in *configurationXML*. The argument must be a well-formed XML string, not a reference to a file. The XML format used is the same as for application defaults. For further details refer to Application Defaults in Chapter 4 – *Service Deployment* in the *Deployment Guide*.
vo.callID ( )	Returns the unique ID of the call as a string of 40 characters.
vo.channel ( )	Returns the code of the channel (voice, video, text, Web) currently used for this service.
vo.databaseType ( )	Returns the type of the database used as VoiceObjects Custom DB logging repository, e.g. *Oracle* or *SQLServer*.
vo.databaseURL ( )	Returns the URL that is required to connect to the VoiceObjects Custom DB logging repository.
vo.dialogID ( )	Returns the unique ID of the dialog as a string of 44 characters.
vo.dialogStepID ( )	Returns the index of the current input state in the dialog, starting with 0. Before the first caller interaction, it returns -1. When Infostore logging on input state level is enabled, the return value matches the entry of column VOLDDSSEQ.DS_STEP for the most recently processed input state.
vo.driver ( )	Returns a string that represents the media platform driver information of the corresponding service (e.g. Nuance_VWS_2_0_VXML_2_0 or IBM_WVS_2_0_VXML_1_0).
vo.driverASREngine ( )	Returns a string that represents the name of the ASR engine (e.g. *Nuance* or *IBM*).
vo.driverID ( )	Returns the ID of the media platform driver of the corresponding service.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions	
Name and Arguments	**Description**
vo.driverProductName ( )	Returns a string that represents the media platform product name of the corresponding service (e.g. *VWS* or *OCSW*).
vo.driverProductVersion ( )	Returns a string that represents the media platform product version of the corresponding service (e.g. 1.2 or 2.0).
vo.driverVendorName ( )	Returns a string that represents the media platform vendor name of the corresponding service (e.g. *Nuance* or *IBM*).
vo.driverVXMLVersion ( )	Returns a string that represents the VoiceXML version of the corresponding service (e.g. 1.0 or 2.0).
vo.errorCode ( )	Returns the error code of the most recently processed Connector item, or zero.
vo.errorMessage ( )	Returns the error message of the most recently processed Connector item, or the empty string.
vo.exitEvent ( )	Returns a string providing the corresponding exit event ending the dialog. Valid return values are: *None, Timeout, Exception, EndOfDialog, CallerHangup, CallerExit,* and *Transfer*. For information on how to exploit this function in dialog end processing, refer to the Module object in this *Object Reference*.
vo.grammarControl ([arg])	When used without an argument, indicates whether grammar-driven application control is activated by default (*true*) or not (*false*). When used with an argument (either *true* or *false*), it sets the default grammar control behavior.
vo.grammarType ([type])	When used without an argument, returns the code of the currently active grammar type (one of *gsl*, *jsgf*, *sgrs_xml*, *precompiled*, *regex*, *abnf*, *none*). When used with an argument, it sets the currently active grammar type to this value. The argument must be one of the grammar type codes listed above. Note that the underlying media platform must be configured appropriately to support the grammar type that is being set.
vo.init (arg)	Resets a variable or collection to its initial value, which was determined when it was first used. Note: If the initial value is not specified, the variable or collection is not reset to an empty string, but to the first value it received during the current call.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions	
Name and Arguments	**Description**
vo.inputMode ([mode])	When used without an argument, returns the code of the currently active input mode (*voice*, *dtmf*, or *voicedtmf*). When used with an argument, it sets the currently active input mode to this value.
vo.language ([languageCode])	When used without an argument, returns the code of the currently active language (e.g. *en-US* or *de-DE*). When used with an argument, it sets the currently active language to this value. Note that the underlying media platform must be configured appropriately to support the language that is being set.
vo.masterDialogID ( )	Returns the unique master dialog ID of the session as a string of 44 characters. The master dialog ID is used when chaining services. If the current service is not part of a chain, this ID is the same as the dialog ID.
vo.md5Sum (txt)	Returns the 128 bit MD5 hash of *txt* as a sequence of 32 hexadecimal digits. Hashing is performed according to RFC 1321, e.g. supplying the string *VoiceObjects* will result in the following hash value: 31414f01914a0366ef9f247bb2eaa9f5.
vo.moduleHistory ([level])	Returns a comma-separated list of reference IDs of Module objects visited so far in the current dialog. When used without an argument the entire list of all visited Module objects will be returned, with the most current Module object at the top of the list. When used with an argument the value of the argument defines how far to go back in the processing history specifying where to start the list (e.g. *level* = 1 means that the list starts with the previously processed Module object instead of the current one). If the specified level exceeds the size of the processing history an empty string is returned. Note: Only Module objects with the option *Enable history tracking* set to *true* are considered.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions	
Name and Arguments	**Description**
vo.moduleRefID ([level])	Returns the reference ID of a Module object processed in the current dialog. When used without an argument the reference ID of the currently processed Module object is returned. When used with an argument (*level*) the reference ID of a parent object of the currently processed Module object is returned. The value of the argument specifies how far to go up in the tree of parent modules (e.g. *level* = 1 means that the reference ID of the first parent of the currently processed Module object is returned). If the specified level exceeds the depth of the module tree an empty string is returned.
vo.moduleSet ([level])	Returns a comma-separated list of reference IDs of Module objects processed so far in the current dialog. The list is sorted alphabetically, and each module only occurs once. When used without an argument the entire list will be returned. When used with an argument the value of the argument defines how far to go back in the processing history specifying where to start the list (e.g. *level* = 1 means that the list starts with the previous Module object instead of the current one). If the specified level exceeds the size of the processing history an empty string is returned. NOTE: Only Module objects with the option *Enable history tracking* set to *true* are considered.
vo.notification (message)	Sends a notification with a predefined ID (1.3.6.1.4.1.24140.1.2.5.8.1) and the specified message.
vo.outputMode ([mode])	When used without an argument, returns the code of the currently active output mode (*audio:tts* or *tts:audio*). When used with an argument, it sets the currently active output mode to this value.
vo.projectInfo (mode)	Provides information on the project that contains the start object of the current service. For *mode* = *project*, it returns the name of the project. For *mode* = *version*, it returns the name of the project version.
vo.pronunciation (var)	Returns the pronunciation pattern associated with the variable provided as first argument.
vo.refresh (arg)	Re-evaluates the initial value of a variable or collection in the current dialog context. For a static initial value, this is the same as INIT(). Note: If the initial value is not specified, the variable or collection is not reset to an empty string, but to the first value it received during the current call.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions	
Name and Arguments	**Description**
vo.recordingScope ([value])	When used without an argument, returns the current value of the corresponding dialog context setting (*nomatch*, *recognition*, or *all*). When used with an argument, it sets the recording scope dialog context setting.
vo.recordUtterances ([value])	When used without an argument, returns the current value of the corresponding dialog context setting (*true* or *false*). When used with an argument, it sets the record utterances dialog context setting.
vo.resetVisitCounter (referenceID)	Resets the visit counter for the object with the specified reference ID. When *all* is given as argument, the visit counters for all objects are reset.
vo.returnCode ()	Returns the HTTP response code for the most recent HTTP or Web Service connector call.
vo.serverContextMappingURL ( )	Returns the complete URL string for the context mapping of the corresponding server instance (e.g. *http://localhost:8099/VoiceObjects/*).
vo.serverHostname ( )	Returns the host name of the corresponding server instance.
vo.serverhostIP ([servletEngine])	Returns the IP address of the corresponding server instance. If the optional parameter *servletEngine* is set to *true* (the default), the address is retrieved from the servlet engine. If set to *false*, it is retrieved from the server configuration.
vo.serverInstanceName ( )	Returns the name of the corresponding server instance.
vo.serverName ( )	Returns the logical name of the corresponding VoiceObjects server.
vo.serverPort ([servletEngine])	Returns the port of the corresponding server instance. If the optional parameter *servletEngine* is set to *true* (the default), the port is retrieved from the servlet engine. If set to *false*, it is retrieved from the server configuration.
vo.serverServletContext ( )	Returns the servlet context of the corresponding server instance (e.g. /VoiceObjects).

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions	
Name and Arguments	**Description**
vo.serverURL ([servletEngine])	Returns the URL leading to the corresponding server instance (e.g. *http://172.22.23.45:8099/VoiceObjects/DialogMapping*). If the optional parameter *servletEngine* is set to *true* (the default), IP and port are retrieved from the servlet engine. If set to *false*, the corresponding values are retrieved from the server configuration.
vo.sessionVariable (varName, [value])	Returns or sets information from the session context. The following arguments are supported: • *ANI* Calling number; same as ANI() • *DNIS* Called number; same as DNIS() • *RDNIS* Redirecting number; not available on all media platforms • *MPSID* Media platform session ID; not available on all media platforms • *AAI* Application to application information; not available on all media platforms • *GCID* Global call ID; not available on all media platforms • *UA* User-agent information; only available for mobile Web driver • *CRMID* Unique caller or customer ID; can be used to join Infostore data with the CRM data warehouse Availability of the information depends on the media platform. In addition, custom arguments can be used.
vo.transferDestination ()	Sets the transfer destination for Infostore. This is particularly useful when transfers are performed using Plug-In objects or Service Chaining.
vo.visitCounter (objRefID)	Returns the number of times the object with the specified reference ID has been visited within the current dialog.

Copyright © 2001-2011 Voxeo Germany GmbH

Dialog Context Functions	
Name and Arguments	**Description**
vo.vsn ()	Returns the name of the VoiceObjects service associated with the current dialog.

Miscellaneous Functions	
Name and Arguments	**Description**
vo.conditional (cond, truevalue, falsevalue)	If *cond* is true, returns *truevalue*. Otherwise, returns *falsevalue*.

For more information about the context functions and their arguments, refer to the Expression object in this *Object Reference*.

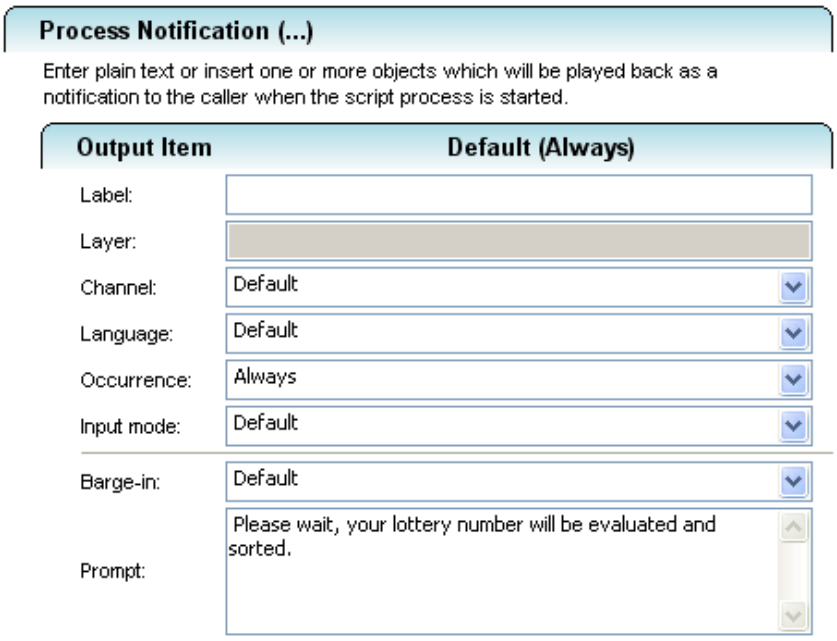
Note: The values returned by these functions are of type *object*. If you want to perform, for example, string operations on the return value of such a function, you need to convert the data type first, e.g. by appending the empty string, as in

```
var lang = vo.language() + '';
```

Process Notification

In the **Process Notification** section, output can be specified that gives the caller a notification that the script process is starting. This parameter is optional.

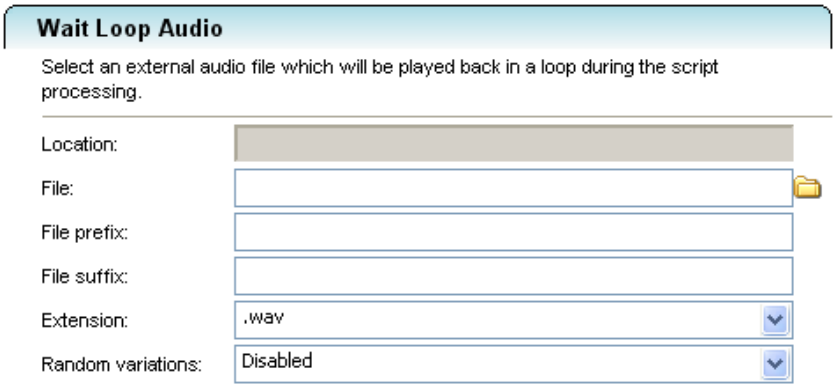
Copyright © 2001-2011 Voxeo Germany GmbH



For further details, refer to the Output object in this *Object Reference*.

Wait Loop Audio

In the **Wait Loop Audio** section, an audio file can be specified that is played back while the script process is running. This parameter is optional. It is typically used for time-consuming Script object operations, e.g. to play hold music.



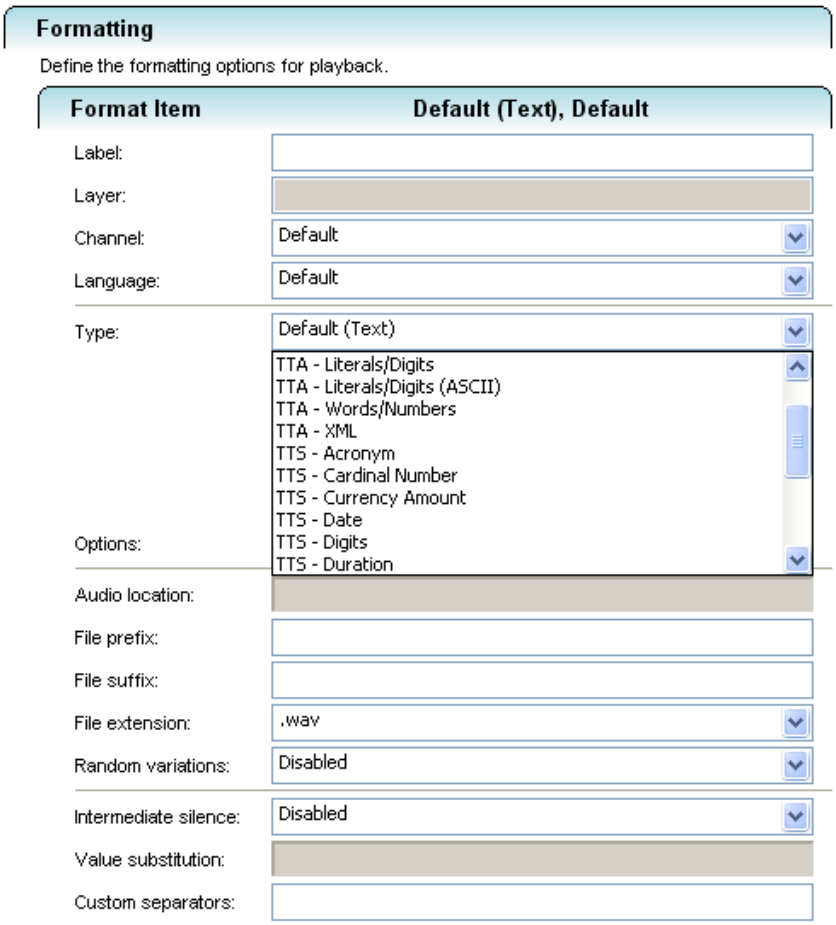
For further details, refer to the Audio object in this *Object Reference*.

Note: On some media platforms, the duration of the audio file must be at least as long as the processing time needed by the Script object. On other platforms, an audio file that is shorter than the processing time required by the script will be looped continuously as needed. For further information on the behavior of audio processing refer to the local documentation of the media platform vendor.

Copyright © 2001-2011 Voxeo Germany GmbH

Formatting

The **Formatting** section defines how the result of the Script object is played back to the caller when it is used within an Output object.



Two different categories of formatting types are offered:

- **Text-To-Audio (TTA)**
  The object content is played back to the caller via the concatenation of prerecorded audio files.
- **Text-To-Speech (TTS)**
  The object content is played back to the caller via text-to-speech synthesis provided by the media platform.

The formatting instructions can be defined as an embedded definition as part of the Script object, or can be linked to an autonomous Format object.

For further details on formatting capabilities refer to the Format object in this *Object Reference.*

Copyright © 2001-2011 Voxeo Germany GmbH

Usage Examples

Usage examples for the Script object are:

- Computing complex formulae that are cumbersome to represent with expressions (e.g. interest rates)
- Evaluating multiple interconnected conditions, and translating them into a finite set of dialog options (which can then be evaluated in a Case object)
- Dynamically creating grammars based on variable content
- Parsing and/or checking validity of data retrieved from back-end systems
- Formatting dynamic content to be played back as a concatenation of audio files (refer to the Format object for more details)

VoiceObjectsXML Definition

The Script object is represented by the VoiceObjectsXML element `<script>`. It has six attributes and nine groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<script>` element uses the embedded `<value>` and `<parameterSet>` elements.

Script

Attributes

- **`location`**
  Defines the location from where the JavaScript file is to be retrieved. Must be a reference to a Resource Locator object.
- **`file`**
  The file name of the JavaScript file. Can be a constant name, or a reference to a an Expression, Script, or Variable object.
- **`embeddedRef`**
  Defines the Script code using a reference to an Expression, Script, or Variable object.
- **`scriptType`**
  Defines the type of scripting language to be used. Legal values include *javascript* and any type that is defined using the Script bus.
  If not specified, defaults to *javascript*.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Script object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Script object.
- **`<value>`**
  Optional embedded constant definition of the script code.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<parameterSet>**
  Defines the parameter set for the Script object.
- **<output>**
  Defines an optional process notification output for the Script object. This child is only relevant if the Script object is used directly within the dialog flow.
- **<audio>**
  Defines an optional wait loop audio for the Script object. This child is only relevant if the Script object is used directly within the dialog flow.
- **<format>**
  Defines the output formatting for the script return value.
- **<eventHandling>**
  Defines the event handling for the Script object.
- **<sequence usage=”postprocessing”>**
  Defines the postprocessing sequence for the Script object.

Example

```
<script name=”Life insurance premium”>
  <value>
    <![CDATA[
        // Compute the life insurance fee based on gender and age
        if (gender == "male")
         {fee = 35;}
        else
         {fee = 27;}

        if (age> 30)
         {fee = fee + (age - 30);}
    ]]>
  </value>
  <parameterSet>
    <item alias=”gender” object=”#Caller gender”/>
    <item alias=”age” object=”#Caller age”/>
    <item alias=”fee” object=”#Insurance premium”/>
  </parameterSet>
</script>
```

Value

Children

- **CDATA**
  Embedded code for a Plug-In or Script object, or embedded value for a Variable object, Collection object, or argument.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<value> 42 </value>


<value>
  <![CDATA[
      // Compute the life insurance fee based on gender and age
      if (gender == "male")
       {fee = 35;}
      else
       {fee = 27;}

      if (age> 30)
       {fee = fee + (age - 30);}
  ]]>
</value>
```

ParameterSet

Children

- **+<item>**
  Defines the list of entries in the parameter set. The use of the `alias` attribute is optional when the `object` attribute is used, and required when the `value` attribute is used.

Example

```
<parameterSet>
  <item alias=”customer” object=”#Customer number”/>
  <item alias=”locale” value=”de-DE”/>
</parameterSet>
```

Item

Attributes

- **object**
  Defines a reference to an object. If this attribute is defined, the `value` attribute must not be defined.
- **value**
  Defines a constant value. If this attribute is defined, the `object` attribute must not be defined.
- **alias**
  Defines the parameter alias.

Example

```
<item alias=”premium” object=”#Annual premium”/>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The Script object is compatible with any other object type and can be used to dynamically control certain properties or preconditions. The following objects reference Script objects most frequently:

Icon	Object Name	Use Case Example
	Expression	A Script object can be used as an argument within an Expression object.
	Connector	A Script object can be used within the parameter set of a Connector object when connecting to an external resource.
	Database	A Script object can be used in the SQL statement definition of a Database object.
	Script	A Script object can be used within the parameter set of another Script object when processing any business logic.
	Plug-In	A Script object can be used within the parameter set when processing a resource of a Plug-In object.
	Log	A Script object can be used in the log statement definition of a Log object.
	Grammar	A Script object can be used to assign a dynamically generated grammar definition to a Grammar object.
	OSDM	A Script object can be linked within the parameter set of any OSDM object.
Any object with a layer or precondition		A Script object can be used anywhere as a layer or precondition.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explaining object names and descriptions.

The Script object name should inform the user what the script code does. The short description should describe which parameters are exchanged, and whether the Script provides a return value. Several examples are listed in the table below:

Name	Description
 Sort Numbers Ascending	Sorts the values of the six variables *Number 1* to *Number 6* in ascending order and assigns the sorted values in that order to variable *Number 1* to *Number 6*. The return value is "*0*" if all numbers are provided and no duplicates exist. Otherwise the return value is a comma-separated list of IDs, which contain duplicates. "*2, 6*" for example means that the value of the variable *Number 2* and *Number 6* are duplicates.

Copyright © 2001-2011 Voxeo Germany GmbH

Name	Description
 Validate Phone Number	Validates if the provided phone number via the variable *Phone Nbr* is complete and correct. The result is *true* if valid; otherwise *false*.
 Calculate Discount Amount	Returns the discounted amount based on the actual shopping cart content.

Copyright © 2001-2011 Voxeo Germany GmbH

Sequence

Overview

The Sequence object is used to arrange objects to be processed sequentially. It is typically used to structure applications into blocks that can be re-used at various places within the respective application or within other applications.

If you want to encapsulate an entire sub-application such as asking callers for their contact information, it is preferable to use a Module object, which can include global settings such as Tuning or Event Handling.

The Object Definition below covers the configuration of the Sequence object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Sequence object belongs to the object category Components.

Dialog Flow Scenario

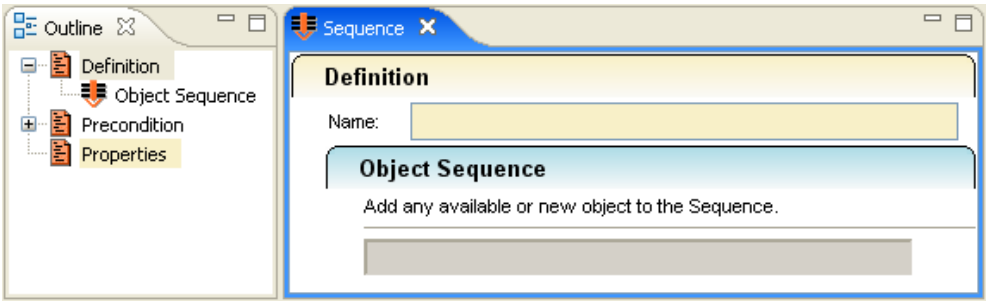
This dialog flow example describes a typical dialog in a restaurant booking application. The caller is asked to select a cuisine (e.g. Italian, Chinese, etc.). The available restaurants are then obtained from a database, and presented to the caller in a browsable list.

Object – Caller	Dialog Flow
	Dialog flow from: *Ask for Cuisine*
	Dialog flow from: *Get Restaurant List*
	Dialog flow from: *Present Browsable List*

Object Definition

The **Definition** of the Sequence object provides one section:

- Object Sequence
  To specify the objects to be processed sequentially.

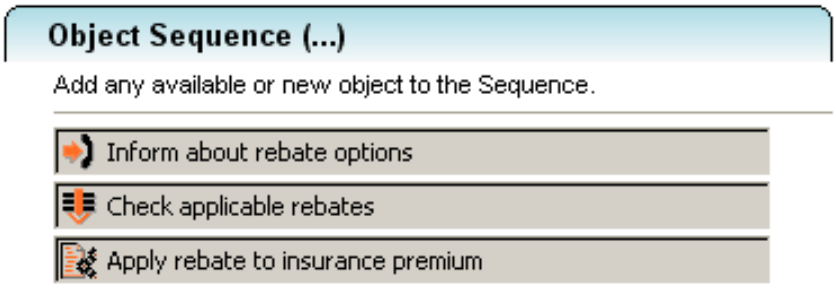


For information regarding additional object configuration, refer to *Precondition* and *Properties* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Object Sequence

The **Object Sequence** section defines the objects to be processed in order of their respective position in the sequence. A Sequence object may contain any number of objects.



VoiceObjectsXML Definition

The Sequence object is represented by the VoiceObjectsXML element `<sequence>`. It has two groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

Sequence

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Sequence object.
- **+(`<module>`, `<input>`, `<output>`, `<sequence>`, `<menu>`, `<confirmation>`, `<list>`, `<osdm>`, `<plugin>`, `<connector>`, `<script>`, `<log>`, `<if>`, `<case>`, `<goto>`, `<loop>`, `<expression>`, `<recording>`, `<transfer>`, `<pause>`, `<exit>`)**
  Defines the list of objects within the Sequence object. The order of elements defines the order of the objects in the Sequence object.

Example

```
<sequence>
  <output>
    <outputItem><text>Please enter your car data
    now.</text></outputItem>
  </output>
  <input link="#Get car info"/>
</sequence>
```

Object Interoperability

The following table contains all object types that can reference a Sequence object:

Copyright © 2001-2011 Voxeo Germany GmbH

Icon	Object Name	Use Case Example
	Module	A Sequence object is always referenced within a Module object –either as an embedded or autonomous object.
	Sequence	A Sequence object can be nested within other Sequence objects.
	Menu	A Sequence object can be used as the destination object within a Menu item.
	Confirmation	A Sequence object can be used as the destination object within a Correction item inside a Confirmation object.
	If	A Sequence object can either be within the THEN item or the ELSE item of an If object.
	Case	A Sequence object can be used in any WHEN item of a Case object.
	Loop	A Sequence object is always referenced within a Loop object –either as an embedded or autonomous object.
	Goto	A Sequence object can be referenced via a Goto object.
	Hyperlink	A Sequence object can be the destination of a Hyperlink object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Sequence object should describe the sequence of steps that are processed. The short description should point out any assumptions or restrictions that apply. The table below lists three examples:

Name	Description
 Process order	Performs the steps necessary to process an order. Requires that a Collection object *Order Items* previously has been filled with the items to be ordered.
 Format back-end data	Prepares the data retrieved from the back-end for presentation to the caller. Required before processing another Sequence object *Present back-end data*.
 Check validity of input	Checks whether the caller input is consistent and conforms to the back-end requirements. Involves both internal computations and back-end access.

Copyright © 2001-2011 Voxeo Germany GmbH

Silence

Overview

The Silence object allows you to play back a silence phase of a specified duration within an Output object. One common use is to provide the proper cadence, or speaking rhythm, when concatenating various resources such as prerecorded audio files, Variable objects, text elements and other Output objects.

Due to its nature the Silence object is only relevant for the voice and video channel; in the text and Web channel it will not be processed. For further information on channels refer to Chapter 10 – *How to Support Multiple Phone Channels* in the *Design Guide*.

The Object Definition below covers the configuration of the Silence object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Silence object belongs to the object category Resources.

Dialog Flow Scenario

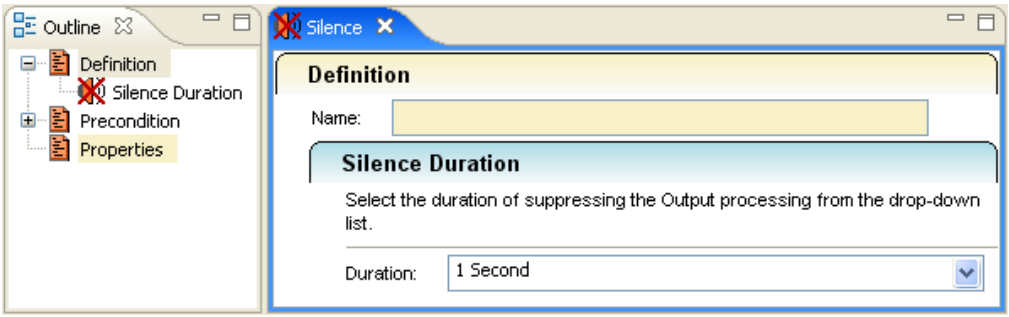
The following dialog flow example demonstrates an Output object with an embedded Audio object, Silence object and plain text.

Object		Call Flow
Output	Audio	[Music jingle]
	Silence	[Silence of 1 second]
	Text	*Hello and welcome to the Voice Portal.*

Object Definition

The **Definition** of the Silence object provides one section:

- Silence Duration

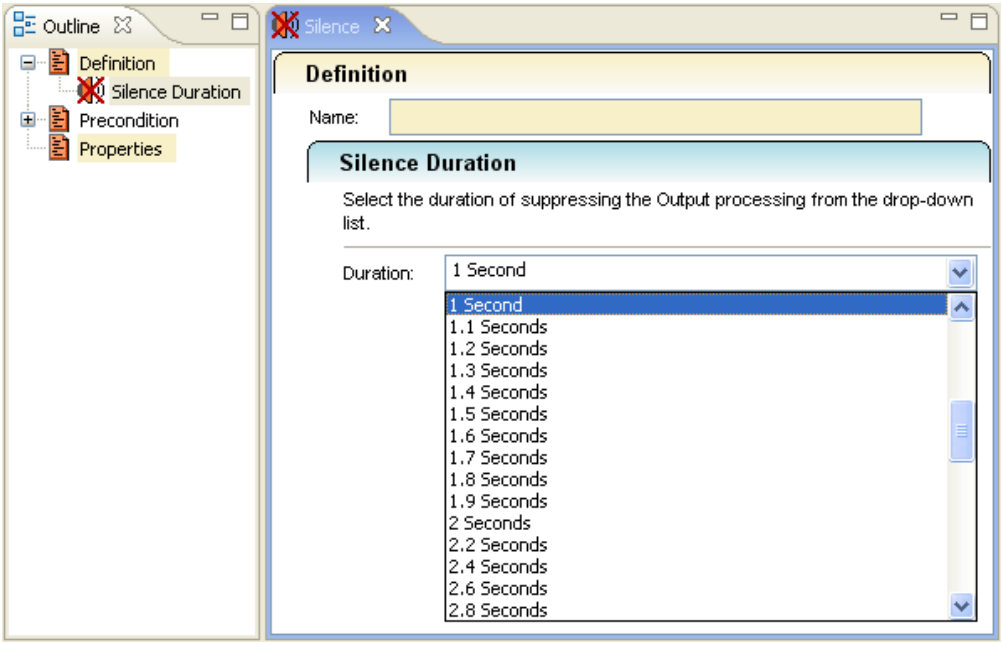


For information regarding additional object configuration, refer to *Precondition* and *Properties* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Silence Duration

In the **Duration** field, the length of the silence phase is defined. The drop-down list offers a set of predefined timeframes. Alternatively a Variable, Expression, Script, or Layer object can be used to specify the silence phase at call time. The value needs to be numeric and will be interpreted as seconds.



Advanced Configuration

Silence in outputs is typically generated on the media platform by use of a text-to-speech (TTS) engine. Therefore on some media platforms, if no TTS engine is installed, the Silence object is not usable in its default configuration.

To enable the use of the Silence object in these cases, it can be configured to use audio files. Note that in this case all durations are rounded down to full 100ms periods, so e.g. a quarter of a second will actually result in 200ms of silence.

To enable the use of audio files, locate the appropriate media platform driver in the file *../VoiceObjects/Platform/WEB-INF/config/MPDrivers.xml*. In the configuration section of the driver, change the setting for `<breakAudioSupport>` to *true*. If you would like to use audio files other than those provided with the VoiceObjects installation, configure their location using the setting `<breakAudioSource>`.

Note that this configuration is transparent to the application itself and only influences how the silence is realized on the individual media platform.

VoiceObjectsXML Definition

The Silence object is represented by the VoiceObjectsXML element `<silence>`. It has one attribute and one group of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

Copyright © 2001-2011 Voxeo Germany GmbH

Silence

Attributes

- **`duration`** [required]
  Defines the duration of the silence, as a numerical value interpreted as seconds. May be static text or a reference to an Expression, Script, or Variable object.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Silence object.

Example

```
<silence duration="2"/>
```

Object Interoperability

The following table contains all object types that can reference a Silence object:

Icon	Object Name	Use Case Example
	Output	The Silence object can be referenced within all objects that provide an embedded output definition.
	OSDM	The Silence object can be linked in the parameter set within any OSDM object.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Silence object should contain information about the duration of the silence phase. When using a Silence object within an output definition the user can immediately identify the length of the silence interval. The table below lists three examples:

Name	Description
1 Second	Interval of 1 second.
Break of 500 ms	Interval of 500 milliseconds.
Dynamic (1-3s)	Silence phase is dynamically defined through the value of a variable.

Copyright © 2001-2011 Voxeo Germany GmbH

Text

Overview

The Text object refers to text files that can be played back or presented through an Output object. It is possible to refer to a single file only, to a specific entry within a file containing multiple text entries, or to a set of files that hold variations of a prompt (for more information on randomizing, see Output).

As examples:

File	Content
welcome.txt	Welcome to the Prime Telecom self-service portal.
selfService. properties	welcome=Welcome to the Prime Telecom self-service portal. mainMenu=Here is the list of choices topup=Top up your pre-paid minutes support=Get help from our support team

In addition, a text generation engine can be referred to through a Text object, in order to embed dynamically generated text into an application.
Alternative text can be provided, which is used if the text files are not available.

While the Text object is provided primarily as a means of managing prompts within text-based applications using the text or Web channel, it can equally be used to play Text-To-Speech prompts in the voice or video channel.

The Object Definition below covers the configuration of the Text object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

For information on how to manage the prompts of your application, refer to the *Storyboard Manager Guide*.

The Text object belongs to the object category Resources.

Dialog Flow Scenario

The following dialog flow example demonstrates an Output object with two Text objects.

Object		Dialog Flow
		*Hello and welcome.*
		*This is your mobile self-service portal.*

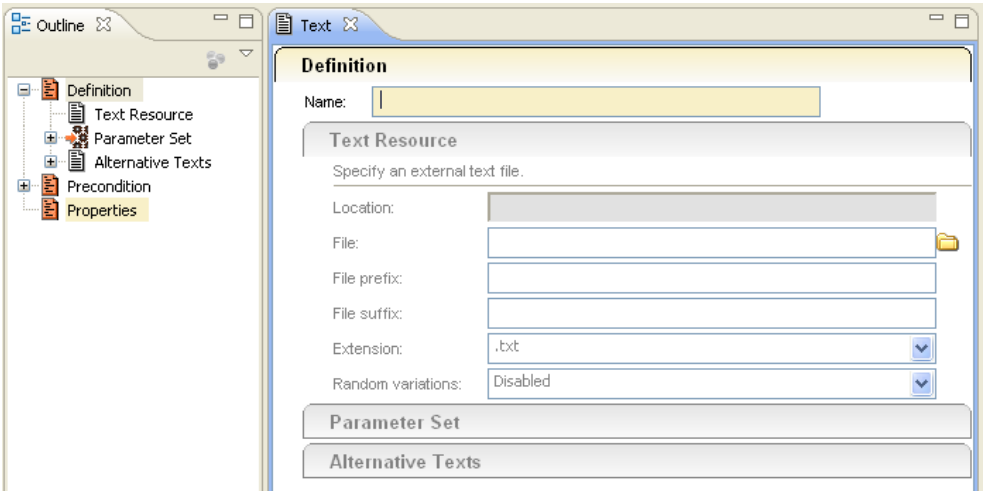
Object Definition

The **Definition** of the Text object provides the following sections:

- Text Resource
  To specify an external text file.

Copyright © 2001-2011 Voxeo Germany GmbH

- Parameter Set
  To specify an optional set of parameters to be passed to text generation engines.
- Alternative Texts
  To specify the alternative text that is used if the text file is not available.



For further details regarding additional object configuration refer to *Precondition* and *Properties* in this *Object Reference*.

Text Resource

The **Text Resource** section provides the following properties:

Property	Description
Location	Optional parameter to specify the location (using a Resource Locator object) of the file to be used.
File	Defines the file name of the file to be used. The name of the file can either be typed in or selected through a file browser by clicking the **Browse** button to the right of the **File** field. The filename can also be provided dynamically during call time by assigning a Variable, Expression, Script, or Layer object. In Desktop for Web the **Browse** button will only be displayed if the Resource Locator object is available and includes a physical path definition.
File prefix	Optionally specifies a prefix that is applied to the file name. The file prefix can also be provided dynamically during call time by assigning a Variable, Expression, Script, or Layer object.
File suffix	Optionally specifies a suffix that is applied to the file name. The file suffix can also be provided dynamically during call time by assigning a Variable, Expression, Script, or Layer object.
Random variations	Defines whether randomizing is enabled for the text file. To enable randomizing set the **Random variations** property to a number between 2 or 99, by selecting an appropriate value from the drop-down list. If randomizing is enabled and different text files are provided with the specified file name extended by

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
	different numbers (e.g. *welcome1.txt*, *welcome2.txt*, *welcome3.txt*), the server will randomly pick one of the available files with a random file extension between 1 and the selected number whenever the Text object is processed. True randomizing ensures that all available text files are played before a repetition occurs. By default, the **Random variations** property is set to *Disabled* indicating that no randomization is used. For more information on randomization see the Output object.

Extension	Specifies the text file extension. Instead of selecting one of the provided file extensions from the drop-down list, the extension can also be supplied by using a Variable, Expression, Script, or Layer object.
	The complete filename for the text file is generated at call time according to the following formula:
	`[prefix] + file + [suffix] + [random number] + [extension]`
	When using property-style references of the form *file#key* (see description below), the file name is generated as
	`[prefix] + file + [suffix] + [extension]`
	and the property name is generated as:
	`property + [random number]`

When the **Text Resource** section refers to a single text file using the **Location** and **File** properties, e.g. *WelcomeCaller*, the complete content of this file is used. When reading the text file, the encoding specified for VoiceObjects Server is used so that, when configured appropriately, the text file may contain UTF-8 characters.

Alternatively it is possible to refer to an entry within a file containing multiple different texts. This file must then have a key/value pair structure based on Java properties files, e.g.:

```
welcome=Hello and welcome to Prime Telecom Self-Service.
goodbye=Thank you for calling Prime Telecom. Have a nice day.
```

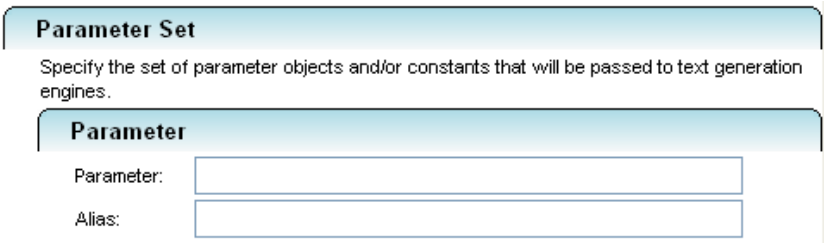
To access a specific key within a file, the notation *textFile#key* is used.
Note that when using this notation, the actual file name and property name are generated according to the formula described above in **Extension**.

VoiceObjects Server caches the content of text files for best performance. The Tuning property **Resource Fetching – Text Max Age** can be used to configure the amount of time (in seconds) that an entry remains active within the cache. When this time expires, the file is loaded again the next time the Text object is processed. When using a file containing multiple text entries in a key/value structure, the entire file is cached or reloaded as necessary.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter Set

The **Parameter Set** section is optional and relevant only for Text objects that refer to external text generation engines. Use it to define any number of parameters, all of which will be added to the final URL as CGI parameters, in order to control the engine if required.



The **Parameter Set** section provides the following properties:

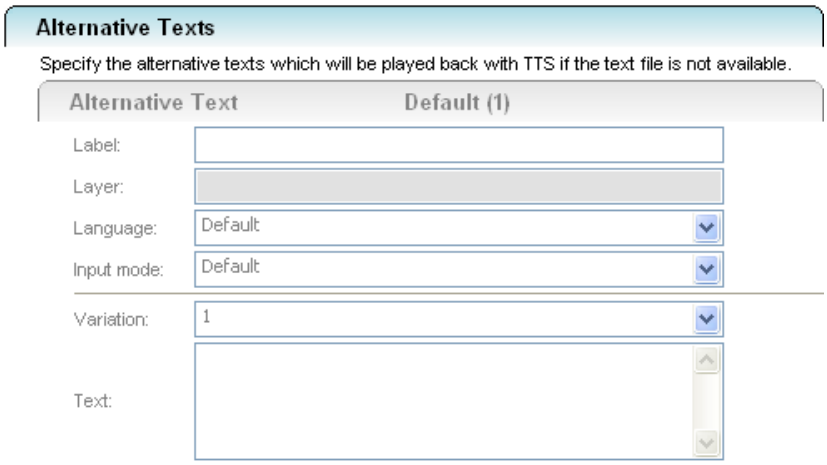
Property	Description
Parameter	Specifies the value of the parameter. Possible parameter types are Variable, Collection, Expression, Layer, Resource Locator or Script objects, and constants. When using a Resource Locator object as a parameter, its URL definition is passed to the script. The field can be left empty to denote an empty value.
Alias	Optionally specifies the name of the parameter. If left empty, the reference ID of the object defined in the **Parameter** field will be used as the name. This means in particular that for constant values an alias needs to be defined (as they do not have a reference ID). Constant parameters without a defined alias are ignored.

The parameters will be added to the final URL in a CGI-string like fashion.

Alternative Texts

The **Alternative Texts** section optionally defines the text that is used when the text file is not defined or cannot be loaded.

Copyright © 2001-2011 Voxeo Germany GmbH



The **Alternative Texts** section contains items to define the various texts for prompt variations, in case the Text object represents more than one file. The text entered into the **Text** field must not contain special characters such as quotation marks, apostrophes, ampersands etc. Also avoid using angle brackets unless you want to include SSML tags into your text. The alternative text can be typed in or can be set at call time by a Variable, Expression, Script, or Layer object.

In addition to the **Text** field, the following properties can be configured:

Property	Description
Label	Optional parameter to identify the alternative text item in a list.
Language	Text objects can represent prompts in multiple languages through dynamic resource locators. The **Language** drop-down list allows you to specify a language for a prompt, so that prompts of different languages can be incorporated into the Text object. When dealing with monolingual applications, this field can be ignored.
Layer	Associates the alternative text item with a custom layer. This item is only eligible for playback if the layer evaluates to *true* at call time. If you reference Text objects in different Output object items using different layer conditions, there is no need to also define the same layers in the Text object itself. If, on the other hand, different outputs are achieved by using a Layer (or Variable, Expression, Script) object within the resource locator definition of the Text object, you can define different alternative texts by associating items with the corresponding layer condition.
Variation	If randomizing is enabled, this field allows you to associate a text with a specific prompt variation number. If the Text object only refers to a single file, this can be left at the default value. Do not define multiple alternative text items with the same variation number (and the other settings being identical), as this may lead to unexpected behavior at call time. If you want to use the same alternative text for all randomized text files, you only need to define an alternative text for variation number 1.

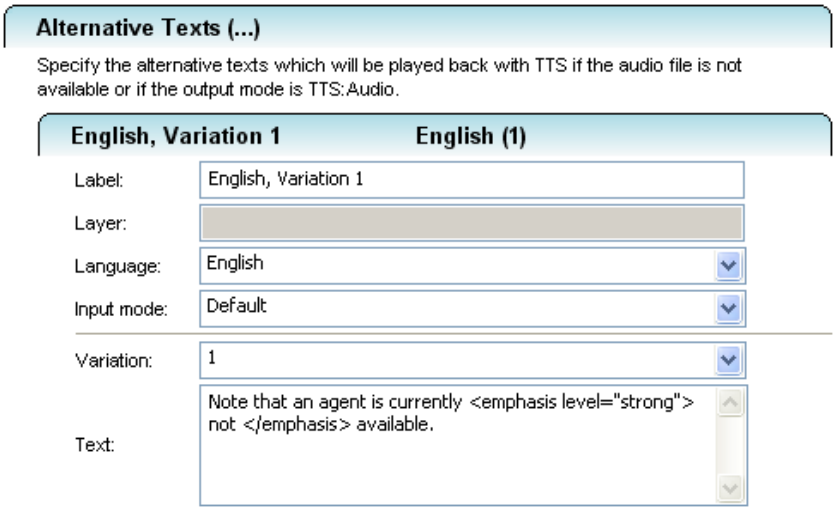
Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Input mode	Associates the alternative text item with a specific input mode. Leave this property at the default value if your application does not distinguish between voice and DTMF input in the wording of your application prompts. Otherwise, tag the item with the appropriate mode.

SSML

The Speech Synthesis Markup Language (SSML), developed by the W3C Voice Browser Working Group, allows adding meta information to text that is supposed to be read out through a speech synthesis system. By using SSML, you have better control over how the TTS engine synthesizes speech. It allows you to define voice-related parameters like volume, speed, pitch, emphasis, pronunciation, etc. For more information on SSML, refer to the specification at the W3C website, or to chapter 4.1.1 of the VoiceXML 2.0 specification.

SSML markup can be included in any alternative text item, as well as in Output items. The server passes the extra code to the media platform without changes. The following example shows how to use SSML to tell the TTS engine to put emphasis on a specific word within a given prompt:



Caution: Since the markup you enclose in the alternative text item definition is passed to the platform as it is, make sure to provide the correct syntax required by SSML. Note that it is up to the platform or TTS engine, respectively, if and how SSML is interpreted. If a TTS engine does not support SSML, it might decide to read the XML tags out aloud.

VoiceObjectsXML Definition

The Text object is represented by the VoiceObjectsXML element `<text>`. It has six attributes and two groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The <text> element uses the embedded <altText> element.

Copyright © 2001-2011 Voxeo Germany GmbH

Text

Attributes

- **`location`**
  Defines the location from where the text file is to be retrieved. Must be a reference to a Resource Locator object.
- **`file`**
  The file name of the text file, without extension. Can be a constant name, or a reference to a Variable, Expression, or Script object.
- **`prefix`**
  A prefix to be used together with the text file name.
- **`suffix`**
  A suffix to be used together with the text file name.
- **`random`**
  Indicates whether randomization should be used. Can be either *disabled* or an integer >= 2 specifying the number of versions in which the file is available. If not specified, defaults to *disabled*.
  For further information on randomization see Random Variations above.
- **`textFileExtension`**
  Specifies the file extension for the text file as either a constant value or a reference to a Variable, Expression, or Script object. Legal constant values are *none, txt, properties*. If not specified, defaults to *txt*.

  The complete filename for the text file is generated at call time according to the following formula:

```
[prefix] + file + [suffix] + [random number] + [extension]
```

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Text object.
- **`?<parameterSet>`**
  Defines the parameter set.
- **`*<altText>`**
  Defines the corresponding alternative text(s).

Example

```
<text location="#Prompt locator" file="welcome"
textFileExtension="txt"/>

<text location="#Prompt locator" file="nomatch"
textFileExtension="txt" random="5"/>
```

Copyright © 2001-2011 Voxeo Germany GmbH

ParameterSet

Children

- **+<item>**
  Defines the list of entries in the parameter set. The use of the `alias` attribute is optional when the `object` attribute is used, and required when the `value` attribute is used.

Example

```
<parameterSet>
  <item alias=”prompt” object=”#Prompt”/>
  <item alias=”speed” value=”0.5”/>
</parameterSet>
```

Item

Attributes

- **object**
  Defines a reference to an object. If this attribute is defined, the `value` attribute must not be defined.
- **value**
  Defines a constant value. If this attribute is defined, the `object` attribute must not be defined.
- **alias**
  Defines the parameter alias.

Example

```
<item alias=”prompt” object=”#Prompt”/>
```

AltText

The `<altText>` element defines the alternative texts to be used for the Text object. It may only occur embedded inside a `<text>` element.

The `<altText>` element has six attributes and contains text as a child.

Attributes

- **label**
  A text string providing a name for the alternative text.
- **random**
  Defines the random index to which this alternative text belongs. An integer >= 1. If not specified, defaults to 1.
- **language**
  Defines the language for which this alternative text is valid. Can be *default* or a valid language code (e.g. *de-DE*, *en-US*, etc.). If not specified, defaults to *default*. Appendix A – *Language Codes* contains a list of all language codes available in VoiceObjects together with the respective language they represent.

Copyright © 2001-2011 Voxeo Germany GmbH

- **`inputMode`**
  Defines the input mode for which this alternative text is valid. Can be *default*, *voice*, *dtmf*, or *voicedtmf*. If not specified, defaults to *default*.
- **`layer`**
  Defines the layer for the alternative text. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **`object`**
  Optional reference to an Expression, Variable, or Script object that defines the alternative text.

Children

- **CDATA**
  The static alternative text itself. Note that the text may not contain special characters such as apostrophes, quotation marks, etc. It may, however, contain XML markup to be used e.g. in combination with SSML.

Example

```
<altText language=”de-DE”>
  Bitte geben Sie Ihre Telefonnummer ein.
</altText>

<altText random=”1”> Sorry? </altText>
<altText random=”2”> Once more, please. </altText>
<altText random=”3”> I didn’t quite get that. </altText>
```

Object Interoperability

The following table contains all object types that can reference a Text object:

Icon	Object Name	Use Case Example
	Output	A Text object can be referenced within an Output object definition. In addition, a Text object can be referenced in all objects that provide Output items.
	List	A Text object can be referenced within the content formatting definition of a List object, to play back as part of the list content.
	OSDM	A Text object can be linked in the parameter set within any OSDM object.

Copyright © 2001-2011 Voxeo Germany GmbH

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Text object should be derived from the name of the Output object it is used in. When used in multiple outputs, its name should provide information about what is being played. The table below lists two examples:

Name	Description
Welcome	Welcome message for the main menu
Generic No Match	Contains five alternatives for a generic No Match prompt (e.g. *Sorry? Your input did not match the expected format.*).

Copyright © 2001-2011 Voxeo Germany GmbH

Transfer

Overview

The Transfer object connects the caller of a service to another entity, referred to below as called party (e.g. a telephone line or another application).

Usage examples of the Transfer object include:

- Voice dialing service:

  - Inbound dialer for an enterprise to automate the main reception

  - Outbound dialer for a telecommunications carrier or an enterprise

- Automated directory assistance with instant transfer option
- Auto-attendant solution with a follow-me transfer option
- Transfer to an operator or a call center agent in any kind of self-service application (e.g. a flight information system)
- Voice dating service with instant call transfer option

Most of the media platforms certified for use with VoiceObjects support two different call transfer modes that can be configured within the Transfer object: bridge transfer and blind transfer. In addition, some platforms also support consultation transfer, which is a special form of blind transfer.

Due to its nature the Transfer object is only motivated for the voice and video channel; in the text and Web channel it will not be processed. For further information on channels refer to Chapter 10 – *How to Support Multiple Phone Channels* in the *Design Guide*.

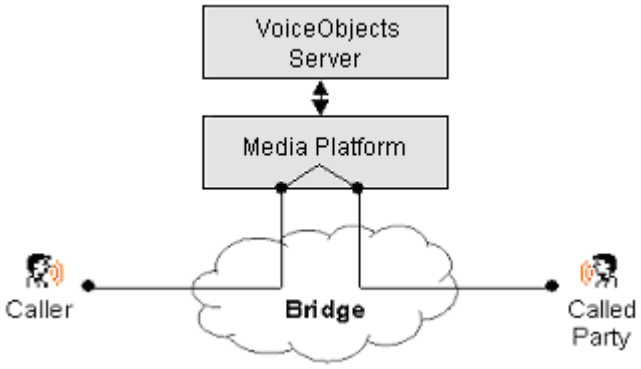
The Object Definition below covers the configuration of the Transfer object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Transfer object belongs to the object category Actions.

Bridge transfer

During a bridge transfer, the media platform connects the caller and the far-end called party in a full duplex conversation, monitoring the progress of the call and optionally listening for speech or DTMF input from the caller (controlled by any included Grammar object). A successful DTMF input or speech recognition will terminate the connection to the called party. If no grammar is specified, the media platform will not listen to input from the caller. The media platform does not monitor in-band signals or speech input from the called party. When the connection between the caller and the called party is terminated, the processing of the dialog flow within VoiceObjects Server resumes. The following figure shows how all parties (caller, called party and media platform) are involved after a successful bridge transfer.

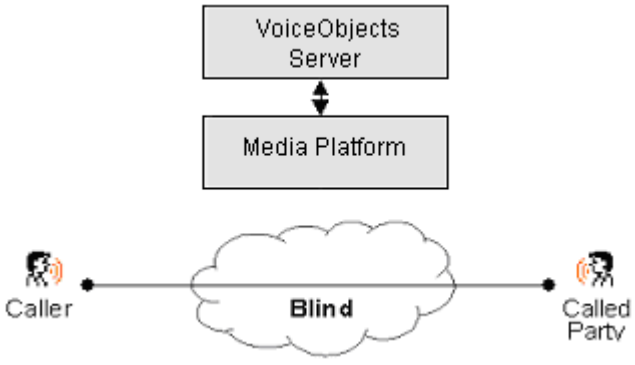
Copyright © 2001-2011 Voxeo Germany GmbH



The bridge transfer allows the configuration of two time intervals. The first one is a timeout interval to establish the connection between the caller and the called party. If the connection timeout is exceeded a no answer event is thrown by the media platform and caught by the Transfer object. The second time interval limits the maximum duration of the established call connection. When this limit is reached the caller and called party are disconnected by the media platform and a maximum time event is thrown to the Transfer object.

Blind transfer

A blind transfer hands off the caller to the telephony network and redirects the caller to the called party. At this point the media platform no longer has access to the call connection, and does not monitor the final outcome. Whether the connection is successful or not, the media platform cannot regain control of the call connection. Still, dialog end processing can take place in VoiceObjects Server immediately after call control has been released. The following figure shows that the media platform and VoiceObjects Server are no longer involved in the call after the transfer has been initiated.



Consultation transfer

When using blind transfer, the outcome of the transfer is not known to the media platform or VoiceObjects Server, so the application cannot continue in case of a transfer failure. If this is desired, though, use consultation transfer. This is basically a blind transfer, so once the transfer is done, both the media platform and VoiceObjects Server terminate their sessions and free resources. But if the transfer cannot be established, call control is returned to the media platform that in turn goes back to VoiceObjects Server. The reason of the transfer failure is stored in a status variable (see below).

Copyright © 2001-2011 Voxeo Germany GmbH

Note: All media platforms used with VoiceObjects Server support blind transfer. Support of bridge and consultation transfer depends on the platform. Media platforms supporting the bridge transfer mode may optionally support DTMF input and/or speech recognition during the call transfer to drop the far-end connection (see the corresponding media platform documentation for further information).

CCXML transfer

The CCXML transfer mode is used to transfer a session back and forth between a VoiceXML and a Call Control XML (CCXML) session. This is useful for certain types of applications such a conferencing in which call control-related activities (e.g. setting up additional call legs) intertwine with dialog flow-related activities (e.g. playing prompts or selecting actions from menus).

A CCXML transfer differs from the other transfer types in that the Transfer object itself does not establish a connection to a called party, but merely hands control over the call back to a pre-existing CCXML session, which then decides how to proceed.

Similar to the situation in a bridged transfer, the VoiceObjects session remains active while the call is handled within the CCXML application, so that it can resume afterwards.

Call transfer fundamentals

When deciding between using a bridge or a blind/consultation transfer, it should always be considered whether or not to allow the caller to resume the interaction with the phone application after the call transfer has been completed. If so, a bridge transfer must be used. If not, a blind or consultation transfer should be performed.

You can use a bridge transfer even if the phone application does nothing more than disconnect the caller when the transfer has ended. However, using a blind transfer in this case minimizes costs by reducing the number of minutes the caller would otherwise consume resources on both the media platform network and the call destination network. When a blind transfer is processed, the call session on the media platform network terminates, and the caller no longer ties up any telephony resources.

If for some reason the outcome of the call transfer is relevant to the application - whether it was answered, the line was busy, the transfer failed, the call duration was so many minutes long, etc. - a bridge or consultation transfer should be used.

Call transfers can fail for several reasons, which are outlined below.

Potential failures when using blind, bridge, or consultation mode:

- Callers attempt to call a destination for which they have no authorization (e.g. a long distance call or a premium service line).
- The specified destination is invalid or incomplete.
- The media platform cannot place a call to the specified destination.

Potential failure only when using the bridge mode:

- The media platform does not have the necessary resources available to place the bridge transfer.

It is possible to minimize call transfer failures by estimating the application-specific call transfer requirements before building and deploying the application; e.g. by ensuring that within an automated call center application the call center infrastructure has sufficient call handling capacity. In addition, it can be useful to take advantage of call center integration capabilities offered by several certified media platforms of VoiceObjects. Some of these media platforms offer the functionality to ensure that a

Copyright © 2001-2011 Voxeo Germany GmbH

call route to an agent is available for the caller before the call transfer gets processed within the phone application.

Handle bridge transfer events

A bridge transfer consists of two processing phases:

- Before a connection to the called party is established.
- After a connection to the called party has been established.

In either processing phase, depending upon the action taken by the caller, the called party or the telephony network, the media platform will invoke a specific exception event, which is available to the Transfer object for further processing within VoiceObjects Server.

While attempting to connect to the called party, the media platform continuously checks call progress indicators. For the duration of a successful bridge transfer, the media platform monitors several telephony events, such as disconnect, on both call legs.

The following table lists the possible results for a bridge transfer before a connection with the called party is established:

Possible Result Prior Connection	Status Code
The caller hung up the phone.	`near_end_disconnect`
The caller forced a call disconnect via a DTMF or speech command.	`near_end_disconnect`
The far-end line of the called party was busy.	`busy`
An intermediate telephony network refused the call connection to the called party.	`network_busy`
The call connection was not answered by the called party within the specified timeout invervall.	`noanswer`
The transfer ended but the reason is not known.	`unknown`

The next table lists the possible outcomes for a bridge transfer after a successful connection between the caller and the called party has been established:

Possible Result After Connection	Status Code
The caller hung up the phone.	`near_end_disconnect`
The caller forced a call disconnect via a DTMF or speech command.	`near_end_disconnect`
The call connection was automaticallly terminated by the media platform because the call duration exceeded the allowed maximum.	`maxtime_disconnect`
The telephony network disconnected the called party from the media platform.	`network_disconnect`
The called party hung up the phone.	`far_end_disconnect`

Copyright © 2001-2011 Voxeo Germany GmbH

Possible Result After Connection	Status Code
The call transfer ended but the status is unknown.	`unknown`

Handle consultation transfer events

If a transfer does not succeed, consultation transfer returns to the application. The outcome is then available with the following status codes:

Possible Result Prior Connection	Status Code
The caller hung up the phone.	`near_end_disconnect`
The caller forced a call disconnect via a DTMF or speech command.	`near_end_disconnect`
The far-end line of the called party was busy.	`busy`
An intermediate telephony network refused the call connection to the called party.	`network_busy`
The call connection was not answered by the called party within the specified timeout invervall.	`noanswer`
The transfer ended but the reason is not known.	`unknown`

Dialog Flow Scenario

The first dialog flow example shows the use of the Transfer object within a voice portal application to establish a bridge connection to an operator for assistance. The caller of the *Voice Portal* is using the service for the very first time and is using operator assistance immediately. After clarification of the questions the call connection is terminated by the operator, and the call continues with the dialog flow of the main menu.

Object – Caller/Called Party	Dialog Flow
	[ Music jingle] *Hello and welcome to the Voice Portal.*
	*Main menu! You have the following options …* *If you need any personal assistance just say operator at any time.*
 Caller	*Operator*
 (Bridged)	*One moment please. We will transfer your call to one of our operators.*
 Called Party	*Good morning!* *My name is Susan, how can I help you?*
 Caller	*Hi! I'm using this service for the first time and I would like to know how to …*

Copyright © 2001-2011 Voxeo Germany GmbH

Object – Caller/Called Party	Dialog Flow
Caller ..... Called Party	< Ongoing conversation >
Called Party	*I'm now transferring you back to the Voice Portal. Goodbye* < Operator hangs up >*.*
	You are back in the main menu! You can choose between …

The next example is based on an outbound voice dialing application. It demonstrates how successive call connections in bridge mode are initiated through the caller within the same dialog flow. The application is using a database system to manage the individual caller profiles and the corresponding personal phone book information.

Object – Caller/Called Party	Dialog Flow
	< Connecting to the database system to load the caller profile and the personal phone book based on the supplied ANI (Automated Number Identification) >
	Good morning Jeff! *Welcome to your personal Voice Dialer.*
	Please say or key in your four-digit PIN.
Caller	[DTMF input] 4 7 1 1
Menu	*OK. What can I do for you?*
Caller	*Call Alex in his office.*
(Bridge)	*One moment!* *I will transfer you to Alex Smith in the Office.* [Ring] … [Ring] … … [Ring]
	Alex Smith is not answering the phone. *Should I transfer you to the mobile phone?*
Caller	*Sure!*
(Bridge)	*Just a moment!* [Ring] … [Ring]
Called Party	*This is Alex.*
Caller	*Hey Alex, this is Jeff. What's your …*
Caller ..... Called Party	< Ongoing conversation with a final hang-up by the called party >

Copyright © 2001-2011 Voxeo Germany GmbH

Object – Caller/Called Party	Dialog Flow
	*Jeff, do you want to call someone else?*
 Caller	*Yes, call Lisa at home. …*

The third dialog flow example demonstrates the use of a blind transfer which hands off the caller to the telephony network and redirects the call to an operator for personal assistance. This happens after the caller has failed to enter his phone number for the third time in a row.

Object – Caller/Called Party	Dialog Flow
	*Enter your ten-digit phone number.*
 Caller	< No input >
	*I'm sorry, I need your phone number.*
 Caller	*8 4 5 3 3 3 3*
	*That's not a complete phone number.* *Please enter your ten-digit phone number including the area code.*
 Caller	*7 0 3 8 4 5 3 3 3*
	*I'm sorry, but I still don't have your complete phone number.*
 (Blind)	*I'll connect you with one of our operators who will assist you.* < Dialog session ends >
 Operator Caller	< Ongoing conversation between the operator and the caller >

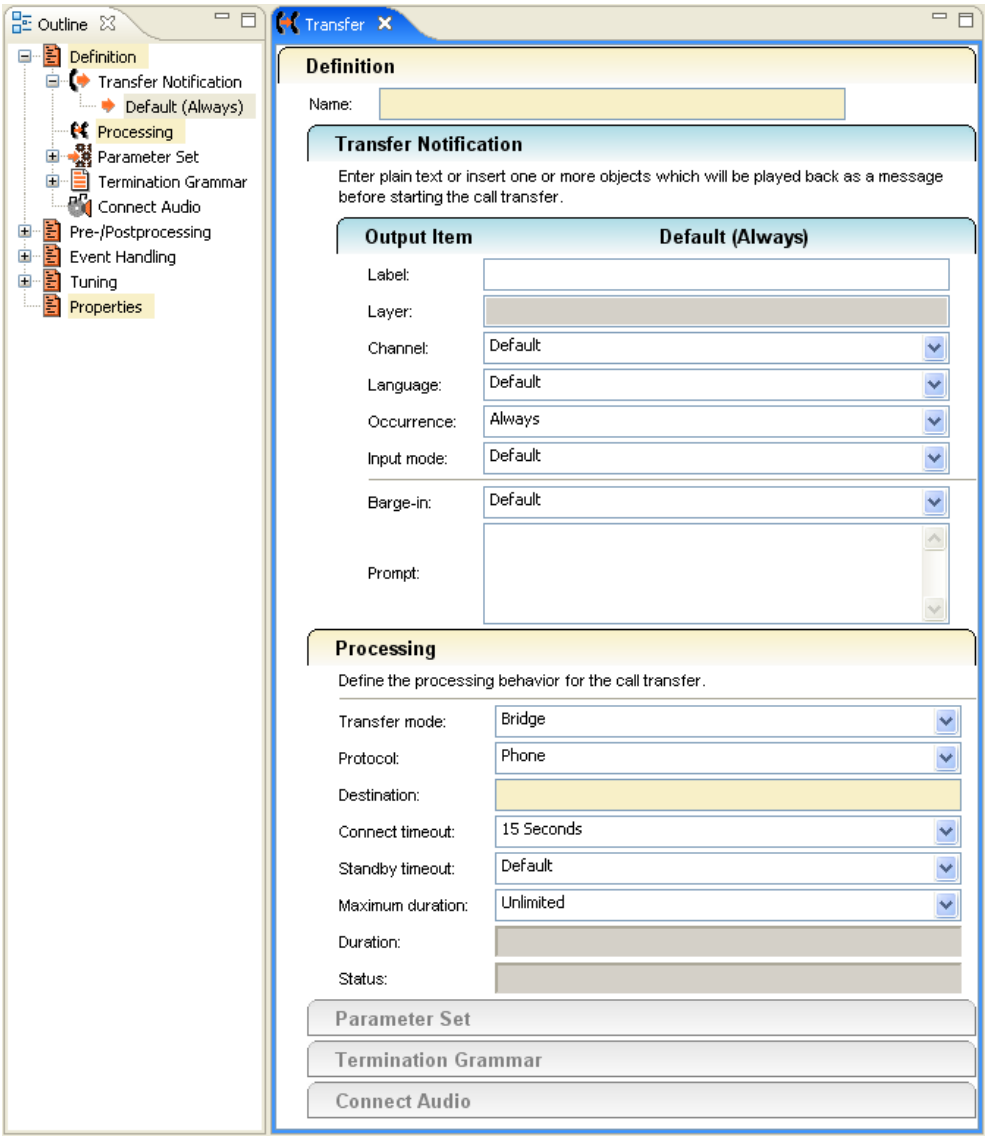
Object Definition

The **Definition** of the Transfer object provides the following sections:

- Transfer Notification
  To inform the caller about the transfer process.
- Processing
  To configure the transfer operation behavior.
- Parameter Set
  To optionally transmit custom parameters with the transfer request.
- Termination Grammar
  To allow the caller to control the connection termination.

Copyright © 2001-2011 Voxeo Germany GmbH

- Connect Audio
  To optionally play back an audio file while connecting.

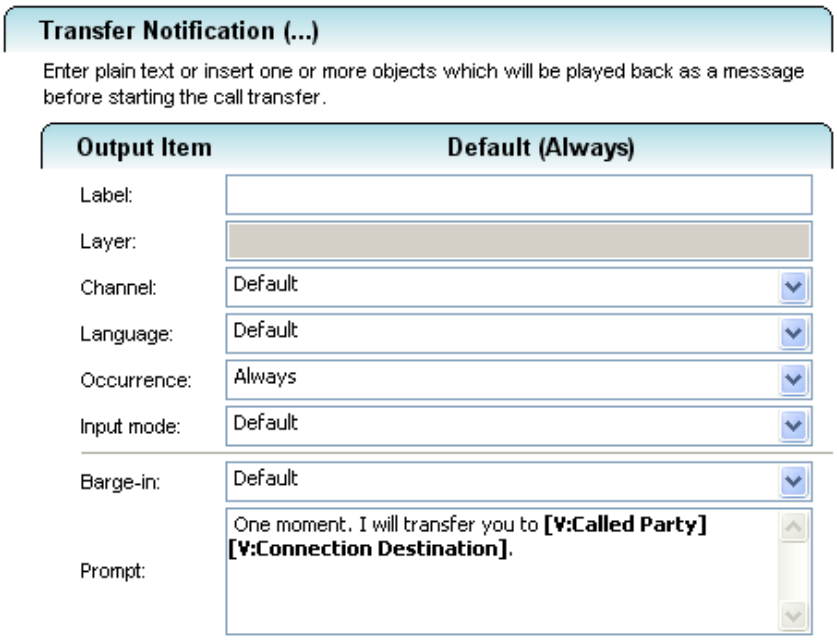


For information regarding additional object configuration refer to *Pre-/Postprocessing*, *Event Handling*, *Tuning*, and *Properties* in this *Object Reference*.

Transfer Notification

The optional output is played back when the Transfer object is entered to notify the caller that a connection to the desired far-end will be initiated. The output should also tell the caller which speech or DTMF command can optionally interrupt the transfer processing via the termination grammar, to return to the original dialog flow. If necessary, timing constraints such as the connect timeout or the maximum connection duration should be mentioned as well.

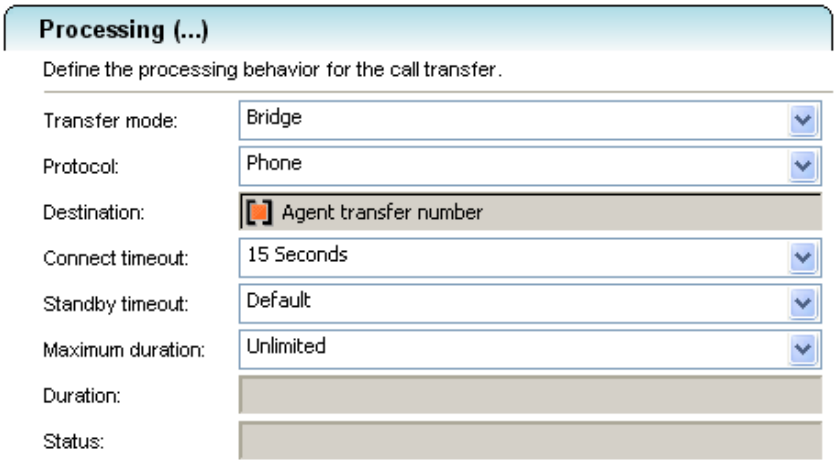
Copyright © 2001-2011 Voxeo Germany GmbH



For more details on output definitions refer to the Output object in this *Object Reference*.

Processing

The **Processing** section specifies mandatory settings, time limits, and optional variables to catch the transfer processing results.



The first part specifies the following mandatory properties:

Property	Description
Transfer mode	Defines the transfer mode as *Bridge*, *Blind*, or *Consultation*. By default it is set to *Bridge*. The transfer mode can also be provided dynamically at call time by assigning a Variable, Expression, Script, or Layer object, which must contain the string `bridge`, `blind`, or `consultation`.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Protocol	Defines a connection protocol to be used for the transfer. The options are *Phone*, *SIP (Session Initiation Protocol)* or *Custom*, where *Phone* is the default setting to configure a regular phone number in the **Destination** field. All media platforms support the phone protocol. Support of the Session Initiation Protocol (SIP) depends on the underlying media platform (details are available in the corresponding media platform documentation). When setting the protocol property to *Custom*, a complete URL for the transfer destination has to be provided in the **Destination** field (see details below).
Destination	Usually contains a regular phone number (e.g. 18001234567) when using the phone protocol, or a SIP address (IP telephony address) when using the Session Initiation Protocol. The connection destination can be provided as a constant value or as the value of a Variable, Expression, Script, or Layer object at call time. If the protocol property is set to *Custom*, the URL for the destination address is based on the URL addressing schemes supported by the individual media platform (see the corresponding media platform documentation for further information). All media platforms usually support the “tel:” URL syntax described in the document URLs for Telephone Calls of the IETF Network Working Group (see http://www.ietf.org/rfc/rfc2806.txt, The Internet Engineering Task Force, April 2000) and may support other URL-based addressing schemes.

The second part of the **Processing** section specifies time limits:

Property	Description
Connect timeout	Defines the time interval to wait while trying to connect to the called party before returning the `noanswer` status. The value is a time designation provided in seconds. The timeout setting only applies if the bridge or consultation transfer mode is enabled, otherwise it will be ignored. By default, the value for the connect timeout is set to 15 seconds. When using a Variable, Expression, Script, or Layer object to define the connect timeout at call time, or when typing in a custom value instead of selecting one of the values from the drop-down list, the value needs to be a positive number and will be applied as seconds, e.g. for a custom timeout of 5.5 minutes, type in 330.
Standby timeout	Defines the maximum time the caller may remain in the transferred call before the server internally terminates the dialog session. If a transfer takes longer than this time interval and the caller wants to return to the dialog after a bridged or CCXML transfer, the server will respond with a "service unavailable" message and make the platform hang up. When using a Variable, Expression, Script, or Layer

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
	object to define the standby timeout at call time, or when typing in a custom value instead of selecting one of the values from the drop-down list, the value needs to be a positive number and will be applied as seconds, e.g. for a custom timeout of 5.5 minutes, type in 330. The default setting *Default* indicates that this timeout is defined within the Service object. For more details on services, refer to *Configuring Servers and Services* in Chapter 2 of the *Deployment Guide*.
Maximum duration	Specifies the time interval that the call connection to the called party is allowed to last. The value is also a time designation provided in seconds and only applies if bridge mode is activated. To enter a custom value, which is interpreted as seconds, enter "15.0" for instance. The default value is set to unlimited which is internally represented by the value 0 (zero). When using a Variable, Expression, Script, or Layer object to define the maximum duration dynamically at call time, the corresponding value needs to be numerical and will be applied as seconds. **Tip:** If the caller exceeds the time interval defined in the **Maximum duration** field, the event *ASR – Max Processing Time* is activated in case a corresponding event handler is defined. A typical use case would be an event handler that first tells the caller that the transfer was interrupted due to the time limit (by linking an Output object in the handler) and then restarts the transfer (by setting **Continuation** to *Return*).

The third part of the **Processing** section references optional variables to catch the transfer processing results for the connection duration and status.

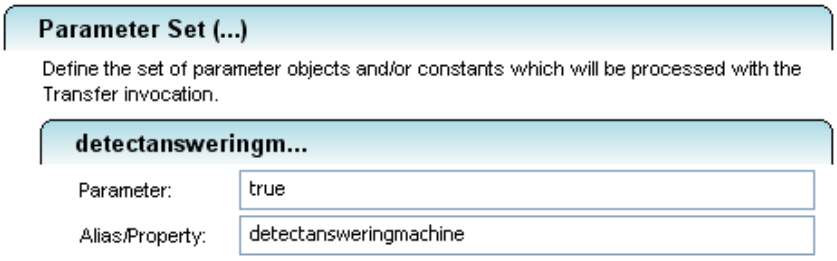
Property	Description
Duration	Optionally references a Variable object to catch the transfer duration time in seconds provided by the media platform. The duration has the value 0 (zero) if a call attempt was terminated by the caller (using an active speech or DTMF command) prior to being answered by the called party. Note that this value is only available in the case of a bridge or consultation transfer. In addition to using this field, you can also access the duration by using the LASTRESULT function through an Expression object.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Status	Optionally references a Variable object to catch the transfer status as documented in the introduction Handle bridge transfer events. The following list shows the set of possible status values in alphabetical order: • `busy` • `far_end_disconnect` • `maxtime_disconnect` • `near_end_disconnect` • `network_busy` • `network_disconnect` • `noanswer` • `unknown` The value of an assigned status variable can be evaluated (e.g. with a Case object) after the Transfer object processing has ended to decide how to continue with the dialog flow. Note that this value is only available in case of a bridge or consultation transfer. In addition to using this field, you can also access the status by using the LASTRESULT function through an Expression object.

Parameter Set

The **Parameter Set** section defines an optional list of parameters that are passed to the media platform. These parameters allow designers to make use of special extensions to the transfer functionality that are provided by a number of media platforms.



The example shown here applies to the VoiceGenie media platform and indicates that the platform should try to detect whether an answering machine answers the transferred call.

The parameters defined in the **Parameter Set** section are placed into the markup as attributes of the `<transfer>` tag, so the VoiceXML code for the example above looks like this:

```
<transfer detectansweringmachine="true"> … </transfer>
```

When using the **Transfer mode** *CCXML*, the following parameters are automatically transmitted without having to define them in the parameter set:

Copyright © 2001-2011 Voxeo Germany GmbH

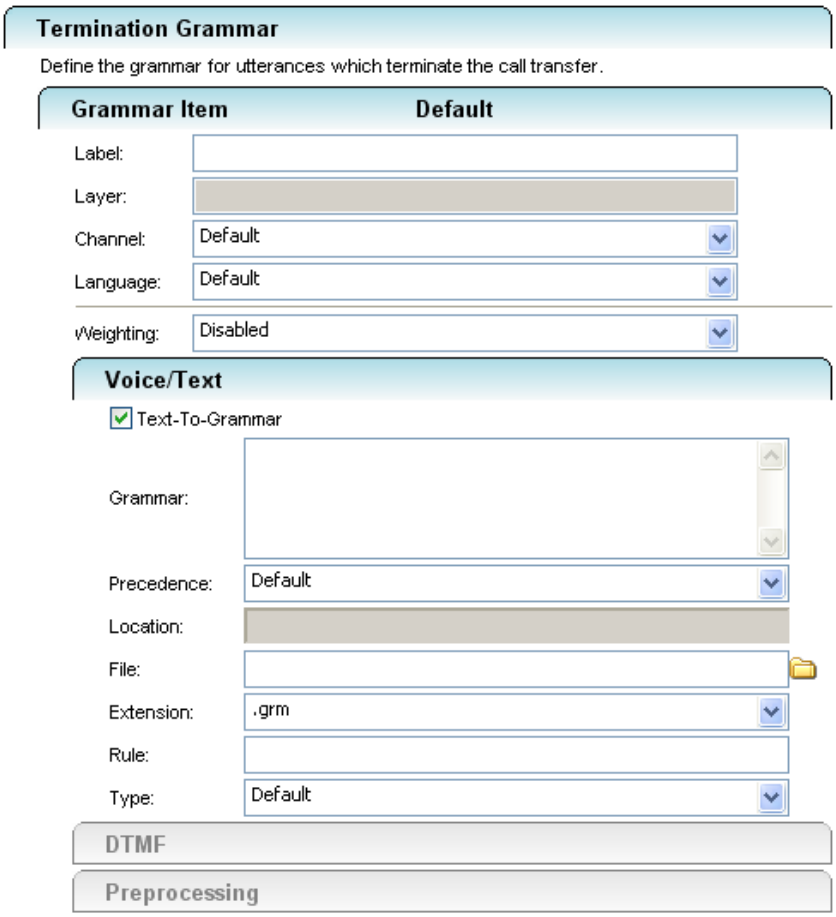
- *VSN*
  The VoiceObjects Service Name (VSN) of the current service.
- *dialogID*
  The dialog ID of the current dialog session.
- *returnURL*
  The URL under which the current dialog session may be continued.

The CCXML application can use these parameters to re-enter the dialog session after having performed any relevant call control activities.

By use of the parameters *goto*=<*ReferenceID*> or *gosub*=<*ReferenceID*> the CCXML application can invoke a different part of the VoiceObjects application when returning, either replacing the normal continuation of the dialog flow (in case of *goto*) or inserting a sub-dialog before continuing with the object following the Transfer (in case of *gosub*).

Termination Grammar

The termination grammar defines the utterances the caller can say and/or the DTMF keys the caller can press to terminate the connection with the called party.



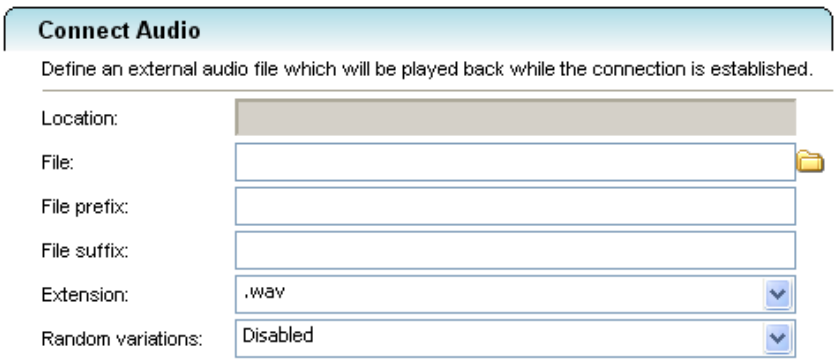
For details on a valid grammar definition refer to the Grammar object in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Note: This feature is only supported for bridged transfers and is not available on all media platforms. See the corresponding media platform documentation for further information.
If a transfer has been terminated through a voice or DTMF command, the utterance and input mode of this command can be retrieved by using the LASTRESULT function. See the Expression object in this *Object Reference* for details.

Connect Audio

The **Connect Audio** section allows configuring an audio source to be played back while the transfer attempt is in progress (before the called party has answered). This is only possible if **Transfer mode** is set to either *Bridge* or *Consultation*.



For detailed information on the fields provided in the **Connect Audio** section refer to the Audio object in this *Object Reference*.

If for any reason the audio file resource cannot be fetched at call time from the media platform (e.g. the file is not available or corrupted), the error will be ignored and the call transfer operation will continue. What the caller will hear instead of the audio resource during the connect interval is media platform and network configuration dependent (e.g. the standard ring tone from the telephony network).

Note: Not all media platforms certified for VoiceObjects support the playback of an audio file while attempting to establish the connection to the called party (see the corresponding media platform documentation for further information).

VoiceObjectsXML Definition

The Transfer object is represented by the VoiceObjectsXML element `<transfer>`. It has eight attributes and seven groups of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<transfer>` element uses the embedded `<parameterSet>` element.

Transfer

Attributes

- **`mode`**
  Defines the mode of the transfer, one of *bridge*, *blind*, or *consultation*. May be a constant value, or a reference to an Expression, Script, or Variable object. If not specified, defaults to *bridge*.

Copyright © 2001-2011 Voxeo Germany GmbH

- **`protocol`**
  Defines the protocol used for the transfer. One of *phone*, *sip*, *custom*. If not specified, defaults to *phone*.
- **`destination`**
  Defines the destination for the transfer. May be a constant value, or a reference to an Expression, Script, or Variable object.
- **`connectTimeout`**
  Defines the timeout for trying to reach the third party. An integer value that is interpreted as seconds. If not specified, defaults to 15.
  May be static text or a reference to an Expression, Script, or Variable object.
  When importing an XDK application into VoiceObjects Desktop, this value must be one of those presented in the Desktop drop-down list (5, 10, 15, 20, 30, 40, 50, 60, 120, 180, 240, 300), or an object reference.
- **`standbyTimeout`**
  Defines the standby timeout for the Transfer object. Either *default* or an integer value that is interpreted as seconds. If not specified, defaults to *default*.
  May be static text or a reference to an Expression, Script, or Variable object.
- **`maxDuration`**
  Defines the maximum duration of the transferred call. Either *unlimited* or an integer value interpreted as seconds. If not specified, defaults to *unlimited*.
  May be static text or a reference to an Expression, Script, or Variable object.
  For details see Maximum Duration above
  When importing an XDK application into VoiceObjects Desktop, this value must be one of those presented in the Desktop drop-down list (60, 120, 180, 240, 300, 600, 900, 1200, 1800, 2400, 3000, 3600), or an object reference.
- **`status`**
  Reference to a Variable object that receives the return status (only relevant for bridged transfers).
- **`duration`**
  Reference to a Variable object that receives the duration of the transferred call in seconds.

Children

- **`<expression usage="precondition">`** or
  `<variable usage="precondition">` or
  `<collection usage="precondition">` or
  `<script usage="precondition">`
  Defines the precondition for the Transfer object.
- **`<sequence usage="preprocessing">`**
  Defines the preprocessing sequence for the Transfer object.
- **`<output>`**
  Defines the transfer notification.
- **`<parameterSet>`**
  Defines the parameter set for the Transfer object.
- **`<grammar>`**
  Defines the grammar used to return to the original application.
- **`<audio>`**
  Defines the connect audio.

Copyright © 2001-2011 Voxeo Germany GmbH

- **<eventHandling>**
  Defines the event handling for the Transfer object.
- **<tuning>**
  Defines the tuning settings for the Transfer object.
- **<sequence usage="postprocessing">**
  Defines the postprocessing sequence for the Transfer object.

Example

```
<transfer mode="blind" destination="+492204845100"/>
```

ParameterSet

Children

- **+<item>**
  Defines the list of entries in the parameter set. The use of the *alias* attribute is optional when the *object* attribute is used, and required when the *value* attribute is used.

Example

```
<parameterSet>
  <item alias="customer" object="#Customer number"/>
  <item alias="locale" value="de-DE"/>
</parameterSet>
```

Item

Attributes

- **object**
  Defines a reference to an object. If this attribute is defined, the value attribute may not be defined.
- **value**
  Defines a constant value. If this attribute is defined, the object attribute may not be defined.
- **alias**
  Defines the parameter alias.

Example

```
<item alias="premium" object="#Annual premium"/>


<item alias="locale" value="en-US"/>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The following table contains all the object types that can reference a Transfer object:

Icon	Object Name	Use Case Example
	Sequence	A call can be transferred at a certain point when using the Transfer object within a Sequence object.
	Menu	A Transfer object can be initiated within a Menu object.
	If	A call can be transferred under a certain condition in an If object.
	Case	A call can be transferred under a certain condition in a Case object.
	Goto	The Transfer object can be referenced within a Goto object.
	Hyperlink	A Hyperlink object is a typical use case where the Transfer object gets activated on demand by the caller through a voice command as described in the first dialog flow scenario.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of the Transfer object should describe in brief what kind of call transfer will be processed and to which destination. The name may contain additional information on the maximum allowed transfer duration or other processing options. The attached description should provide further details how the Transfer object is processed. See the examples below:

Name	Description
Transfer to Operator (Blind)	Transfers the call in blind mode to the call center operator.
Connect to SIP Destination	Connects the caller to the specified destination via the SIP protocol in bridge mode. The destination is specified by the variable *SIP Destination*.
Transfer to VoiceBanking	Transfers the call to the 3rd party phone application VoiceBanking (1-800-VOICE-BANK).

Copyright © 2001-2011 Voxeo Germany GmbH

Tuning

Overview

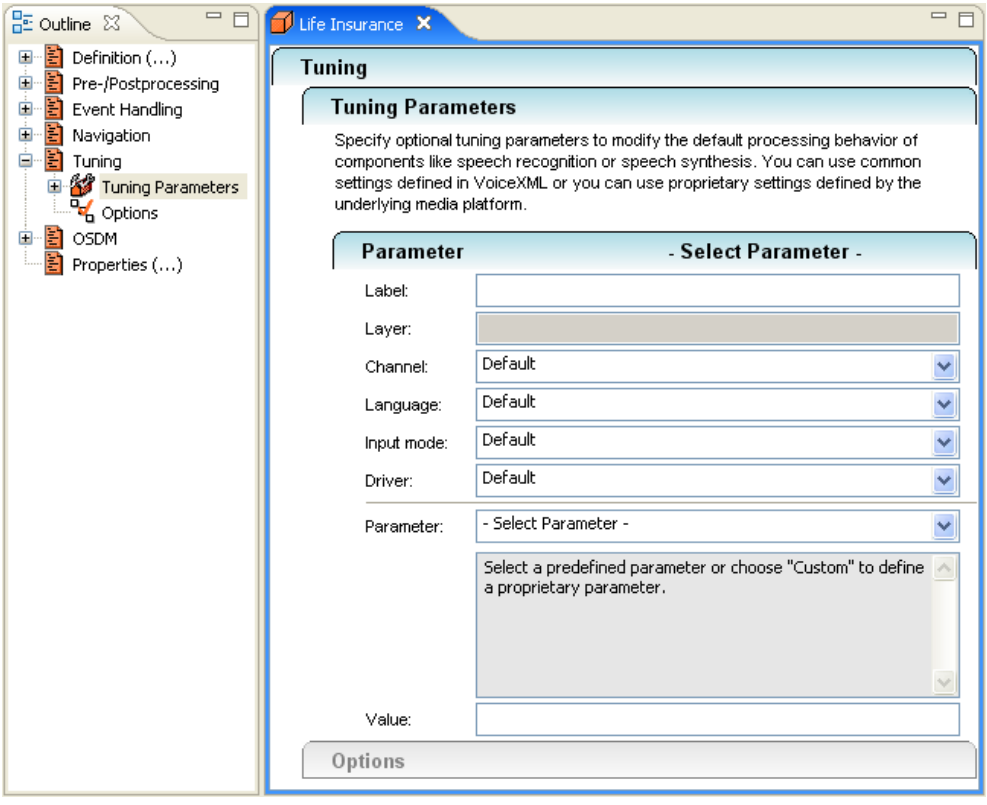
Tuning parameters modify the default processing behavior of media platform components like speech recognition or speech synthesis. You can use common settings defined in VoiceXML or you can use proprietary settings of the media platform, both for applications running in the voice or video channel. In addition, some VoiceObjects specific parameters exist that apply to the text and Web channel mainly. Each setting can be dynamically set at call time by specifying layer settings for language, input mode, media platform driver, channel, or by defining a custom layer.

Tuning parameters defined in Module objects may be inherited by the objects that are part of the Module object's sequence (see Inheritance).

The paragraph Tuning Definition describes how to define tuning using VoiceObjects Desktop. For information on how to define it using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph. The table of available predefined tuning parameters covers both VoiceObjects Desktop and VoiceObjectsXML notation.

Tuning Definition

Tuning parameters can be defined in the **Tuning** section of the respective object editors (entered by clicking it in the **Outline** in Desktop for Eclipse; represented by a tab in Desktop for Web).



Copyright © 2001-2011 Voxeo Germany GmbH

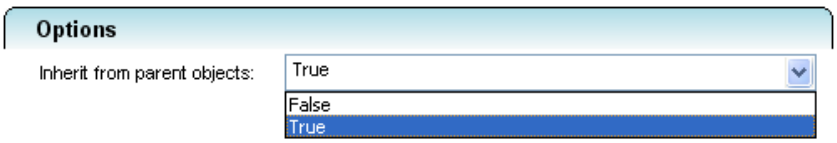
For each **Parameter** you can specify the following properties:

Property	Description
Label	Optional setting to identify the tuning parameter in a list.
Layer	Specifies a layer via a Variable, Expression, Layer, or Script object to dynamically enable or disable the tuning parameter.
Channel	Specifies a channel as an optional filter condition for the parameter.
Language	Specifies a language as an optional filter condition for the parameter.
Input mode	Specifies an input mode as an optional filter condition for the parameter.
Driver	Specifies a media platform driver as an optional filter condition for the parameter.
Parameter	Choose a predefined parameter or select *Custom* to define a custom or proprietary parameter. If you choose a predefined parameter its description will be displayed in the text field below (respectively to the right in Desktop for Web). The description holds the name of the parameter followed by the actual tuning property written into the VoiceXML code (in parentheses), for applications run in the voice or video channel. All parameters apply to the voice and video channel only, unless stated otherwise. For a list of all available predefined parameters see Tuning Parameters below. If you choose *Custom* in the **Parameter** field, enter a custom parameter in the **Custom** field (displayed dynamically in Desktop for Eclipse). This can be done by either typing it in or specifying it via a Variable, Expression, Layer, or Script object.
Value	A valid value for the parameter. This can be provided by either typing it in or specifying it via a Variable, Expression, Layer, or Script object.

Options

Tuning parameters defined in Module objects are inherited by the objects that are part of the Module object's sequence.

Inheritance can be disabled (*False*) or enabled (*True*, the default) within the **Options** section.



Copyright © 2001-2011 Voxeo Germany GmbH

VoiceObjectsXML Definition

Tuning parameters are defined using the `<tuning>` element. It provides the **`inheritance`** attribute indicating whether tuning settings from a parent object should be used (*true*, the default) or not (*false*). In addition, it uses one or more `<tuningItem>` elements as children to define the individual tuning settings.

The `<tuningItem>` element provides the following seven attributes:

- **label**
  A text string providing a name for the tuning item.
- **parameter** [required]
  The name of the tuning parameter to be set. Can either be one of the predefined parameters or a custom parameter.
  For a list of all available predefined parameters see Tuning Parameters below.
- **value** [required]
  The value for the tuning parameter. May be a constant value or a reference to an Expression, Script, or Variable object.
- **language**
  Defines the relevant language for the tuning item. Can be *default* or a valid language code (e.g. *de-DE*, *en-US*, etc.). If not specified, defaults to *default*. Appendix A – Language Codes contains a list of all language codes available in VoiceObjects together with the respective language they represent.
- **inputMode**
  Defines the input mode for which this tuning item is valid. Can be *default*, *voice*, *dtmf*, or *voicedtmf*. If not specified, defaults to *default*.
- **layer**
  Defines the layer for the tuning item. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **driver**
  Defines the media platform driver for which the tuning setting is applicable. The list is provided in the Document Type Definition (DTD).
  If not specified, defaults to *default*.
- **channel**
  Defines the channel for which the tuning setting is applicable. Can be *voice*, *video*, *text*, *web*, *voiceVideo*, or *textWeb*. If not specified, defaults to *default*.

Tuning Parameters

The following table lists all available predefined tuning parameters with a description of what you can use them for.

The *Parameter* column mentions the names as listed in VoiceObjects Desktop, followed by the names used in VoiceObjectsXML.

In the *Description* column you can find the actual tuning property written into the VoiceXML code (in parentheses), for applications run in the voice or video channel.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter	Description
Infostore – Input State Name VoiceObjectsXML: INFInputStateName	The parameter *Input State Name* optionally defines the name under which an input state is logged in Infostore. By default, the name of the object generating the input state is used.
Input – Barge-in Type VoiceObjectsXML: ASRBargeinType	The parameter *Barge-in Type* (bargeintype) indicates what kind of input from the caller can interrupt the playback of a prompt or audio output. Possible values for this parameter are dependent on the media platform used. Example values for the VoiceGenie media platform are: e*nergy* - any noise can barge in on a prompt; *speech* (default value) - any caller utterance can barge in on a prompt; *recognition* - only barges in on a prompt when the caller input matches a grammar.
Input – Complete Timeout VoiceObjectsXML: ASRCompleteTimeout	The parameter *Complete Timeout* (completetimeout) specifies the amount of time to wait for additional input from the caller after the speech recognition engine has recognized speech input matching one of the input grammars. A long complete timeout value delays the recognition completion and therefore makes the application response slow. A short complete timeout may lead to a grammar utterance being broken up inappropriately. Reasonable complete timeout values are typically in the range of *0.3* to *1.5* seconds. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *0.5s* or *700ms*). The default value is media platform dependent.
Input – Confidence Level VoiceObjectsXML: ASRConfidenceLevel	The parameter *Confidence Level* (confidencelevel) is specified with a float value in the range of *0.0* (minimum confidence) to *1.0* (maximum confidence). Speech recognition results are rejected when the recognition confidence is below the specified threshold. Setting this parameter higher will result in fewer false recognitions at the cost of more No Match events. Setting it lower will reduce the number of No Match events at the risk of more false recognitions. The default value is *0.5*.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter	Description
Input – DTMF Inter-Digit Timeout VoiceObjectsXML: DTMFInterDigitTimeout	The parameter *DTMF Inter-Digit Timeout* (interdigittimeout) is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *3s* or *2500ms*). It controls the amount of time that the speech recognition engine waits for another DTMF input before it decides that the caller has finished entering digits and returns a recognition or a No Match event. The default value is media platform dependent.
Input – DTMF Termination Key VoiceObjectsXML: DTMFTermChar	The parameter *DTMF Termination Key* specifies the (optional) terminating character for DTMF recognition. If this parameter is set, the caller can finalize the DTMF input with the specified key press. The default value is the hash character.
Input – DTMF Termination Timeout VoiceObjectsXML: DTMFTermTimeout	The parameter *DTMF Termination Timeout* (termtimeout) is a time designation expressed as an unsigned number by *s* for time in seconds; *ms* for time in milliseconds (e.g. *2s* or1*500ms*). It specifies the terminating timeout to use when recognizing DTMF input. The timeout is activated when the last DTMF input has brought the grammar to a termination point at which no additional DTMF input is allowed by the grammar. If a DTMF termination key is defined, the caller can enter this DTMF key within the given timeout. If the caller fails to enter this optional DTMF key, the DTMF recognition ends and the recognized value is returned. The default timeout value is *0s*.
Input – Incomplete Timeout VoiceObjectsXML: ASRIncompleteTimeout	The parameter *Incomplete Timeout* (incompletetimeout) specifies the amount of time to wait for additional speech input when the caller has begun speaking but the input does not yet match a complete grammar. Once the timeout is activated, the partial recognition result is rejected by the media platform with a No Match event. A long incomplete timeout value delays the recognition completion and therefore makes the application response slow. A short incomplete timeout may lead to an in-grammar utterance being broken up inappropriately. The incomplete timeout is usually longer than the complete timeout to allow the caller to pause mid-utterance (for example, to breathe). Reasonable incomplete timeout values are typically in the range of *0.3* to *1.5* seconds. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *0.5s* or *700ms*). The default value is media platform dependent.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter	Description
Input – Input modes VoiceObjectsXML: ASRInputModes	The parameter *Input Modes* (inputmodes) determines which input modality to use: DTMF and/or voice. Speech recognition is disabled when setting the value to *dtmf*. DTMF recognition is disabled when setting the value to *voice*. Both modes are enabled when using the value *dtmf voice* or *voice dtmf*, which is the default for media platforms that support both input modes.
Input – Max N-Best VoiceObjectsXML: ASRMaxNBest	The parameter *Max N-Best* (maxnbest) controls the maximum number of N-best results returned by the speech recognition engine. It is an integer value greater than or equal to 1. Note that the actual number of N-best results returned by the speech recognition engine may be smaller than the value provided by *Max N-Best*.
Input – Max Speech Duration VoiceObjectsXML: ASRMaxSpeechTimeout	The parameter *Max Speech Duration* (maxspeechtimeout) controls the maximum duration of speech input from the caller. If this time elapses before the caller stops speaking, the event *ASR – Max Speech Duration* is thrown by the media platform. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *6s* or *4500ms*). The default duration is media platform dependent.
Input – Recognition Speed vs. Accuracy VoiceObjectsXML: ASRSpeedVsAccuracy	The *Recognition Speed vs. Accuracy* (speechvsaccuracy) parameter is a float value between *0.0* (fastest recognition) and *1.0* (best accuracy). It lets you control the trade-off between speed and accuracy in the speech recognition engine. Lower values cause potential recognition results with a low probability to be pruned early in the search process, resulting in faster recognition speed. Higher values retain potential recognition results longer, resulting in slower but more accurate recognitions. The default value is *0.5*.
Input – Sensitivity Level VoiceObjectsXML: ASRSensitivity	The parameter *Sensitivity Level* (sensitivity) is specified with a float value in the range of *0.0* (least sensitive to noise) to *1.0* (highly sensitive to quiet input). It controls the sensitivity of the speech recognition process. Higher values will allow the speech recognition engine to recognize softer speech input but will also pick up more background noise. This parameter may need to be tuned downward if the voice application is used in noisy environments, or upward if it is used in very quiet environments. The default value is *0.5*.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter	Description
Input – Speech Timeout VoiceObjectsXML: ASRTimeout	The parameter *Speech Timeout* (timeout) specifies the maximum time the speech recognition engine waits for caller input. After the specified amount of time has lapsed, a No Input event is thrown by the media platform. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *3.1s* or *2400ms*). The default value is set to *3s* at the VoiceObjects media platform driver level.
Input – Universal Commands VoiceObjectsXML: ASRUniversals	The parameter *Universal Commands* (universals) specifies which universal command grammars should be active within the voice dialog. The value is one of: *all* - make all universal grammars active; *none* - deactivate all universal grammars; a space-separated list of the universal grammar names (e.g. *help exit cancel*) - make the specified grammars active and all non-specified grammars inactive. Usually the following three universal commands are supported: *help* - the caller is asking for help; *exit* - the caller is asking to exit the application; *cancel* - the caller is asking to cancel the prompt that is being played. The default value is set to *help* at the VoiceObjects media platform driver level for all media platforms supporting universal commands. Some media platforms may optionally provide platform-specific universal command grammars. For example the Nuance Café media platform does support the command *goback* - the caller wants to retract the last response and go back to an earlier part of the dialog interaction.
Output – TTS Engine VoiceObjectsXML: TTSEngine	The parameter *TTS Engine* tells the media platform which TTS (text-to-speech) engine to use. It is valid to specify a particular engine (e.g. for a specific female or male voice) only if that engine is installed on the media platform being used to run this application. This parameter is media platform specific and is only supported by the following vendors: VoiceGenie, Voxpilot and Nuance Café. For further detail see the relevant media platform documentation.
Presentation – Bottom Logo URL VoiceObjectsXML: PRESBottomLogoURL	The parameter *Bottom Logo URL* only applies to the Web channel. Use it to specify a URL of an image file that will be added to the bottom of the page. By default, there is no Bottom Logo URL defined. Note that in a multi-tenant environment, the *Resource Locator URL* of the site settings is applied to the URL.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter	Description
Presentation – Input VoiceObjectsXML: PRESInput	The parameter *Input* only applies to the text and Web channel. Use it to specify how the grammar of an Input object is to be presented. In the text channel, it can have the values *text* (grammar not displayed/available, free entry of text) or *menu* (grammar displayed as a menu of labeled items). In the Web channel, it can have the values *text* (text area for free input), *password* (text area for free input in password mode), *menu* (grammar displayed as a set of hyperlinks), *radio* (grammar displayed as radio-buttons), *checkbox* (grammar displayed as checkbox), or *list* (grammar displayed as a drop-down list). The default value depends on the number of entries in the grammar. In the text channel, *password* is interpreted as *text*, while *radio* and *list* are interpreted as *menu*.
Presentation – Input Maximum Length VoiceObjectsXML: PRESInputMaxLength	The parameter *Maximum Length* only applies to the Web channel. It specifies the maximum allowed length of input fields, i.e. the number of characters the caller is allowed to type in an input field. By default, no maximum is defined.
Presentation – Menu VoiceObjectsXML: PRESMenu	The parameter *Menu* only applies to the Web channel. Use it to specify how the items of a Menu object are to be displayed. It can have the values *horizontal* (all Menu items in one row), *vertical* (all items below each other), *2columns* (two items per row), *3columns* (three items per row). The default is *vertical*.
Presentation – Proceed Key VoiceObjectsXML: PRESProceedKey	The parameter *Proceed Key* only applies to the text channel. It defines the key that activates the “proceed” button in standalone outputs. No default key is defined, which means that the USSD gateway selects one.
Presentation – Proceed Label VoiceObjectsXML: PRESProceedLabel	The parameter *Proceed Label* only applies to the text and Web channel. It allows localizing the displayed label of the “proceed” button in standalone outputs. The default value is >>.
Presentation – Slot Label VoiceObjectsXML: PRESSlotLabel	The parameter *Slot Label* only applies to the Web channel. It specifies the labels to show in front of Input object slots (if any). Labels can contain text or markup and can be defined per slot, using the following syntax: label1(slot1),label2(slot2), ... By default, no slot labels are defined.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter	Description
Presentation – Style Sheet URL VoiceObjectsXML: PRESCSSURL	The parameter *Style Sheet URL* only applies to the Web channel. Use it to specify a URL pointing to a CSS file that acts as a style sheet for the rendered XHTML page. By default, *Platform\Resources\System\StyleSheet\default.css* is used. Note that in a multi-tenant environment, the *Resource Locator URL* of the site settings is applied to the URL.
Presentation – Top Logo URL VoiceObjectsXML: PRESTopLogoURL	The parameter *Top Logo URL* only applies to the Web channel. Use it to specify a URL of an image file that will be added to the top of the page. By default, there is no Top Logo URL defined. Note that in a multi-tenant environment, the *Resource Locator URL* of the site settings is applied to the URL.
Resource Fetching – Audio Fetch Hint VoiceObjectsXML: RFAudioFetchHint	The parameter *Audio Fetch Hint* (audiofetchhint) tells the media platform whether or not audio files may be pre-fetched. The value is either: *prefetch* (default) - permit, but do not require, the media platform to prefetch audio files; or *safe* - fetch audio files only when they are needed, never before.
Resource Fetching – Audio Max Age VoiceObjectsXML: RFAudioMaxAge	The parameter *Audio Max Age* (audiomaxage) tells the media platform the maximum acceptable age of cached audio files. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *86400s*). An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again. The default value is media platform specific.
Resource Fetching – Audio Fetch Stale VoiceObjectsXML: RFAudioMaxStale	The parameter *Audio Max Stale* (audiomaxstale) tells the media platform the maximum acceptable time during which expired cached audio files can still be used. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *300s*). A cached file that has been expired for more than the maximum stale time will be re-fetched; one that has been stale for less than or equal to the maximum stale time will be used. The default value is media platform specific.
Resource Fetching – Text Max Age VoiceObjectsXML: RFTextMaxAge	The parameter *Text Max Age* tells VoiceObjects Server the maximum acceptable age of cached text files. The value is a time designation expressed as an unsigned number in seconds (e.g. *86400*). An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again. The default value is 86400 seconds, corresponding to one day.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter	Description
Resource Fetching – Document Fetch Hint VoiceObjectsXML: RFDocumentFetchHint	The parameter *Document Fetch Hint* (documentfetchhint) tells the media platform whether or not VXML files may be pre-fetched. The value is either: *prefetch* (default) - permit, but do not require, the media platform to pre-fetch VXML files; or *safe* - fetch VXML files only when they are needed, never before.
Resource Fetching – Document Max Age VoiceObjectsXML: RFDocumentMaxAge	The parameter *Document Max Age* (documentmaxage) tells the media platform the maximum acceptable age of cached VXML files. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *86400s*). An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again. The default value is media platform specific.
Resource Fetching – Document Max Stale VoiceObjectsXML: RFDocumentMaxStale	The parameter *Document Max Stale* (documentmaxstale) tells the media platform the maximum acceptable time during which expired cached VXML files can still be used. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *300s*). A cached file that has been expired for more than the maximum stale time will be re-fetched; one that has been stale for less than or equal to the maximum stale time will be used. The default value is media platform specific.
Resource Fetching – Fetch Timeout VoiceObjectsXML: RFFetchTimeout	The parameter *Fetch Timeout* (fetchtimeout) tells the media platform the general timeout period for resource fetches. The media platform will wait this long for a requested resource to be returned before throwing an “error.badfetch" event. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *60s* or *30000ms*). The default value is media platform dependent.
Resource Fetching – Grammar Fetch Hint VoiceObjectsXML: RFGrammarFetchHint	The parameter *Grammar Fetch Hint* (grammarfetchhint) tells the media platform whether or not grammar files may be pre-fetched. The value is either: *prefetch* (default) - permit, but do not require, the media platform to pre-fetch grammar files; or *safe* - fetch grammar files only when they are needed, never before.
Resource Fetching – Grammar Fetch Options VoiceObjectsXML: RFGrammarFetchOptions	The parameter *Grammar Fetch Options* provides optional additional information to the media platform on how to fetch grammar files. Any number of options may be set separated by blanks in the format *option=value* (without quotes around the value).

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter	Description
	When working with Voxeo Prophecy it can e.g. be used to set *voxeo:useuri=true* in order to bypass grammar conversion.
Resource Fetching – Grammar Max Age VoiceObjectsXML: RFGrammarMaxAge	The parameter *Grammar Max Age* (grammarmaxage) tells the media platform the maximum acceptable age of cached grammar files. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *86400s*). An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again. The default value is media platform specific.
Resource Fetching – Grammar Max Stale VoiceObjectsXML: RFGrammarMaxStale	The parameter *Grammar Max Stale* (grammarmaxstale) tells the media platform the maximum acceptable time during which expired cached grammar files can still be used. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *300s*). A cached file that has been expired for more than the maximum stale time will be re-fetched; one that has been stale for less than or equal to the maximum stale time will be used. The default value is media platform specific.
Resource Fetching – Object Fetch Hint VoiceObjectsXML: RFObjectFetchHint	The parameter *Object Fetch Hint* (objectfetchhint) tells the media platform whether or not files referenced in an `<object>` element may be pre-fetched. The value is either: *prefetch* (default) - permit, but do not require, the media platform to pre-fetch VXML files; or *safe* - fetch VXML files only when they are needed, never before.
Resource Fetching – Object Max Age VoiceObjectsXML: RFObjectMaxAge	The parameter *Object Max Age* (objectmaxage) tells the media platform the maximum acceptable age of cached files referenced in an <object> element. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *86400s*). An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again. The default value is media platform specific.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter	Description
Resource Fetching – Object Max Stale VoiceObjectsXML: RFObjectMaxStale	The parameter *Object Max Stale* (objectmaxstale) tells the media platform the maximum acceptable time during which expired cached files referenced in an `<object>` element can still be used. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *300s*). A cached file that has been expired for more than the maximum stale time will be re-fetched; one that has been stale for less than or equal to the maximum stale time will be used. The default value is media platform specific.
Resource Fetching – Script Fetch Hint VoiceObjectsXML: RFScriptFetchHint	The parameter *Script Fetch Hint* (scriptfetchhint) tells the media platform whether or not VXML files may be pre-fetched. The value is either: *prefetch* (default) - permit, but do not require, the media platform to pre-fetch VXML files; or *safe* - fetch VXML files only when they are needed, never before.
Resource Fetching – Script Max Age VoiceObjectsXML: RFScriptMaxAge	The parameter *Script Max Age* (scriptmaxage) tells the media platform the maximum acceptable age of cached script files. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *86400s*). An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again. The default value is media platform specific.
Resource Fetching – Script Max Stale VoiceObjectsXML: RFScriptMaxStale	The parameter *Script Max Stale* (scriptmaxstale) tells the media platform the maximum acceptable time during which expired cached script files can still be used. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *300s*). A cached file that has been expired for more than the maximum stale time will be re-fetched; one that has been stale for less than or equal to the maximum stale time will be used. The default value is media platform specific.
Resource Fetching – Wait Audio Delay VoiceObjectsXML: RFFetchAudioDelay	The parameter *Wait Audio Delay* (fetchaudiodelay) is the amount of time to wait after a resource download is started by the media platform before the wait audio clip is played in the background. This parameter is useful to keep the period of silence short before the audio clip starts playing. If the resource download completes during the period of silence, the audio clip will not be played at all. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *2s* or *2500ms*). The default value is media platform dependent.

Copyright © 2001-2011 Voxeo Germany GmbH

Parameter	Description
Resource Fetching – Wait Audio Minimum VoiceObjectsXML: RFFetchAudioMinimum	The parameter *Wait Audio Minimum* (fetchaudiominimum) specifies a time interval to play a wait audio clip (see *Wait Audio URL*), once started, even if the resource download completes in the meantime. The idea is that once the caller does begin to hear the wait audio clip, it should not be stopped too quickly. The value is a time designation expressed as an unsigned number followed by *s* for time in seconds; *ms* for time in milliseconds (e.g. *2s* or *2500ms*). The default value is media platform dependent.
Resource Fetching – Wait Audio URL VoiceObjectsXML: RFFetchAudio	The parameter *Wait Audio URL* (fetchaudio) specifies the URL of an audio clip to be played back while waiting for a resource to be fetched by the media platform. The default behavior of most of the media platforms is not to play any audio clip. The playing of the audio clip is governed by the parameters *Wait Audio Delay*, and *Wait Audio Minimum* in effect at the time of the fetch.

Copyright © 2001-2011 Voxeo Germany GmbH

Variable

Overview

The Variable object provides the concept of a non-typed variable as in other standard programming or scripting languages to control the dialog context as well as the business logic of an application.

The Variable object can be used within almost any other object to control certain object properties or processing conditions dynamically at call time. It is also used to present to the caller dynamic application content as part of an Output object. Therefore the Variable object provides the capability to define an output format using either text-to-speech (TTS) processing (e.g. to say the value as a phone number) or alternatively text-to-audio (TTA) processing (e.g. to read back the phone number digit by digit via prerecorded audio resources).

The scope of a Variable object is global to the dialog context of each individual call. A Variable object will be initialized according to its initial value declaration the first time it is used in the dialog flow. By default the initial value is empty (empty string) which is equivalent to NULL. A Variable object containing an empty string can be tested with the Expression function ISNULL ( ), which will return *true*, or ISNOTNULL ( ), which will return *false*, respectively. In a Boolean context the Variable object evaluates to *true* unless its value is an empty string, 0 (zero), or FALSE (in any combination of upper and lower case letters).

A Variable object can also be initialized via the initial URL of the media platform which starts a dialog for the corresponding service. This can be achieved by adding a variable parameter to the URL string with an ampersand sign (&). This parameter is constructed with the prefix `var` followed by the reference ID of the Variable object. Finally the value is added with an equal sign (=).

`...&var`*`VariableID=value`*

The following example shows a complete URL initializing the caller status variable with the reference ID `Status` to the value `gold`:

http://MyServer/VoiceObjects/DialogMapping?VSN=MyService&varStatus=gold

It is possible to initialize multiple variables at the same time as outlined below:

http://MyServer/VoiceObjects/DialogMapping?VSN=MyService&varStatus=gold&
varName=Jeff

The Object Definition below covers the configuration of the Variable object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Variable object belongs to the object category Logic.

Dialog Flow Scenario

The following short dialog flow example as part of a flight information service demonstrates an Output object containing some text and two Variable objects with the terminal and gate information. The variables are played back with text-to-speech (TTS) processing.

Copyright © 2001-2011 Voxeo Germany GmbH

Object		Dialog Flow
	Text	*Your flight will leave on time at terminal:*
		C
	Text	*gate:*
		73

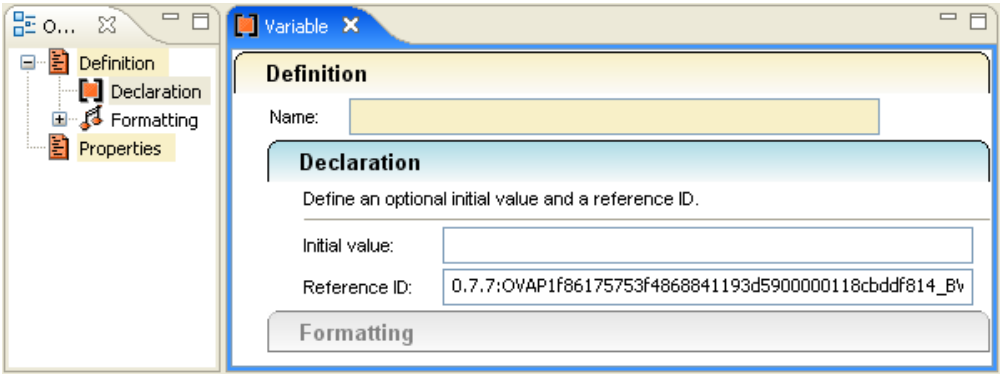
The next dialog flow demonstrates the same scenario as above but using prerecorded audio resources. The Variable objects are played back with text-to-audio (TTA) processing.

Object		Dialog Flow
		Your flight will leave on time at terminal:
		[c.wav] *C*
		gate:
		[73.wav] *seventy three*

Object Definition

The **Definition** of the Variable object provides the following sections:

- Declaration
  To declare the initial value and reference ID.
- Formatting
  To define formatting instructions for playback within an Output.

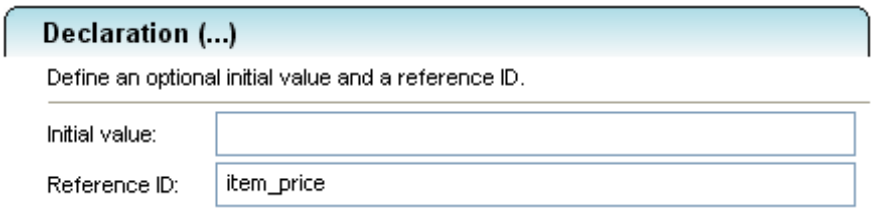


For information regarding the **Properties** section, refer to *Properties* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Declaration

The **Declaration** section provides the **Initial Value** and the **Reference ID** field (which is also available in **Properties**).



The **Initial Value** can be provided as a constant or can be dynamically initialized at call time from the value of other Variable objects, Expression, Layer, or Script objects. The initial value of the Variable object is evaluated when the Variable object is first used within the dialog flow at call time.

You can use the system functions INIT() and REFRESH() to initialize a variable to the initial value again at a later point in the dialog. REFRESH() re-evaluates the initial value in the current dialog context, whereas INIT() resets the variable to the value it got when it was first initialized. For static initial values, both functions have the same effect.

Caution: Make sure that the Variable object itself is not part of its own initial value definition (e.g. by being part of the parameter set passed to the Script object that computes the initial value), as this will lead to infinite loops.

The **Reference ID** can be used to identify and reference a Variable object outside of the VoiceObjects Server framework, e.g. when using the Variable object in conjunction with a Java connector or a VoiceXML plug-in. In these cases, the automatically generated reference ID should be replaced by a more readable name.

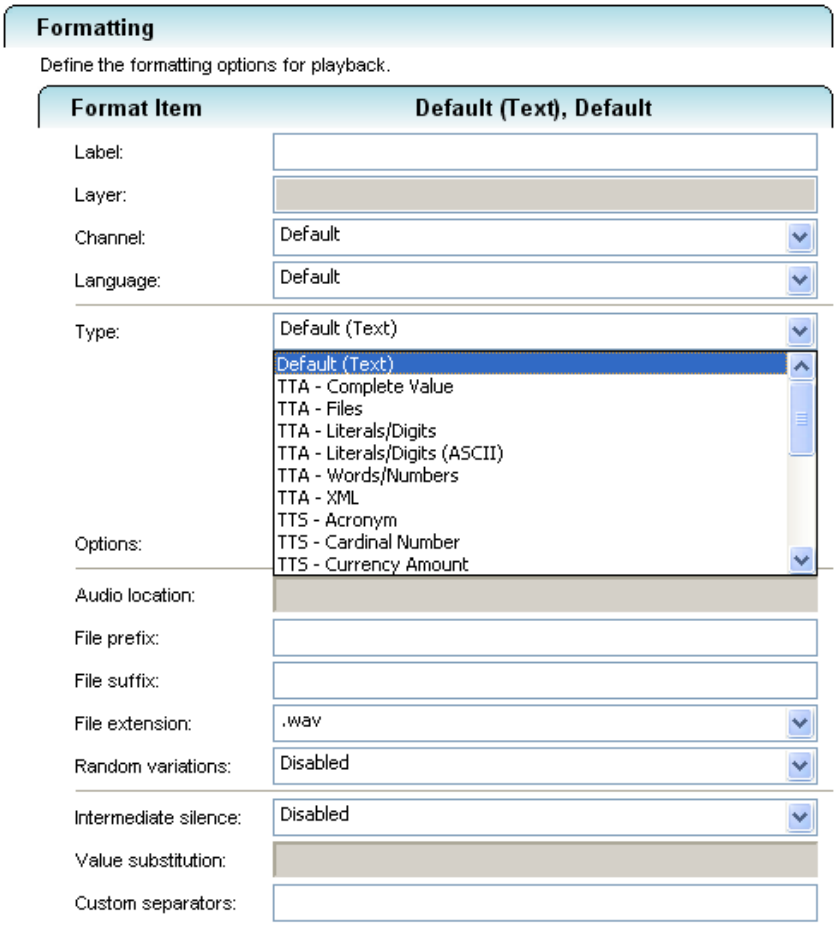
Note that the reference ID may not contain blanks or other special characters except hyphens and underscores. It is not case sensitive and limited to 40 characters. No two objects within one project may have the same reference ID.

In addition to the actual value a Variable object can hold a *pronunciation value*, which can be set by a grammar during speech recognition in the voice or video channel. This value can be retrieved during call time using the PRONUNCIATION function through the Expression object. For more information on associating pronunciation patterns with a Variable object, refer to Input in this *Object Reference*.

Formatting

The **Formatting** section defines how the content of the Variable object is presented to the caller when it is used within an Output object.

Copyright © 2001-2011 Voxeo Germany GmbH



Two different categories of formatting types are offered, which are only relevant for the voice and video channel:

- **Text-To-Audio (TTA)**
  The object content is played back to the caller via the concatenation of prerecorded audio files.
- **Text-To-Speech (TTS)**
  The object content is played back to the caller via text-to-speech synthesis provided by the media platform.

The formatting instructions can be defined as an embedded definition as part of the Variable object, or can be linked to an autonomous Format object.

To define formatting types for the text and Web channel, use the Formatting bus as described in Appendix B – *How to Use the Formatting Bus* in the *Administration Guide*.

For further details on formatting capabilities refer to the Format object in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

VoiceObjectsXML Definition

The Variable object is represented by the VoiceObjectsXML element `<variable>`. It has one attribute and two children.

In addition, the element has the standard attributes described in the *XDK Guide*.



Caution: The *name* attribute is required for Variable objects.

The `<variable>` element uses the embedded `<value>` element.

Variable

Attributes

- **`initial`**
  Reference to an Expression, Script, or Variable object defining the initial value of the variable.

Children

- **`<value>`**
  Optionally defines the constant initial value of the Variable object.
- **`<format>`**
  Defines the output formatting for the Variable object.

Example

```
<variable name=”pi” initial=”#Compute pi”/>


<variable name=”Car Model” referenceID=”carModel”>
  <value> 99 </value>
</variable>
```

Value

Children

- CDATA
  Embedded value.

Example

```
<value> 42 </value>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The Variable object is compatible with any other object type and can be used to dynamically control certain properties or preconditions. The following objects reference Variable objects most frequently:

Icon	Object Name	Use Case Example
	Input	A Variable object can be used to store a recognition result.
	Output	A Variable object can be used for the playback of the variable value to the caller.
	List	A Variable object can be used to store the list selection result.
	Grammar	A Variable object can be used to assign a dynamically generated grammar definition to a Grammar object.
	Connector	A Variable object can be used within the parameter set of a Connector object when connecting to an external resource.
	Database	A Variable object can be used in the SQL statement definition of a Database object.
	Script	A Variable object can be used within the parameter set when processing any business logic using a Script object.
	Plug-In	A Variable object can be used within the parameter set when processing a resource of a Plug-In object.
	Log	A Variable object can be used within the log statement definition of a Log object.
	Expression	An Expression object can be used to assign a new value to a Variable object or to use a variable as an argument.
	Goto	A Variable object can specify the reference ID of the destination object within the Goto object.
	OSDM	A Variable object can be linked within the parameter set of any OSDM object
Any object with a layer or precondition		A Variable object can be used anywhere as a layer or precondition.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Variable object should be as short as possible and at the same time precise enough that other users will know immediately what the variable is used for. The short description of a Variable object should provide additional information about the format of the content (e.g. number, text, date, time or Boolean values), the

Copyright © 2001-2011 Voxeo Germany GmbH

expected or valid value domain and finally something about the initial value (if available). If multiple variables are used regarding the same context they should start with the same prefix (e.g. Customer ID, Customer Name, Customer ZIP, etc.).

The table below lists a few examples as a general guideline:

Name	Description
Phone Number	Ten-digit telephone number.
Customer ID	Alphanumerical 40 character ID of the customer. The initial value is set to *N/A*.
Customer Name	Title, first and last name of the customer (e.g. Mr. George Hill).
Caller Status	Status of the caller which is defined as *silver*, *gold* or *platinum*; otherwise *unknown*.
Frequent Flyer	Contains the value *true* if the caller is a frequent flyer otherwise *false* which is the default value.
Arrival Date	Flight arrival date in the format YYYYMMDD.
Reservation Time	Time designation in the format HH:MM (hours:minutes). Valid values for HH are: 18 to 22. The only valid value for MM is 30.
Shopping Cart Balance	Actual balance of the shopping cart as decimal currency amount with a leading *$* symbol. The initial value is set to *$0.00*.

Copyright © 2001-2011 Voxeo Germany GmbH

Video

Overview

The Video object refers to video files that can be played back through an Output object. Videos are primarily aimed at mobile devices in a 3G network.
The Video object can refer to a single file only, or to a set of files that hold different variations of the video. In addition, a video generation engine can be referred to through a Video object, in order to embed dynamically rendered videos into an application.
The supported file formats depend on the underlying media platform that will do the final video processing, as well as the mobile device that plays back the video.

The Video object can only be used in the video channel.

The Object Definition below covers the configuration of the Video object with VoiceObjects Desktop. For information on how to define this object type using VoiceObjectsXML refer to the VoiceObjectsXML Definition paragraph.

The Video object belongs to the object category Resources.

Dialog Flow Scenario

The following dialog flow demonstrates the use of video messages within a dating application. The Video object is linked in an Output object, together with an Audio object that represents a prompt that is played prior to the video:

Object		Dialog Flow
		[*Okay, I found one matching candidate. Let's give her a chance to introduce herself.*]
		

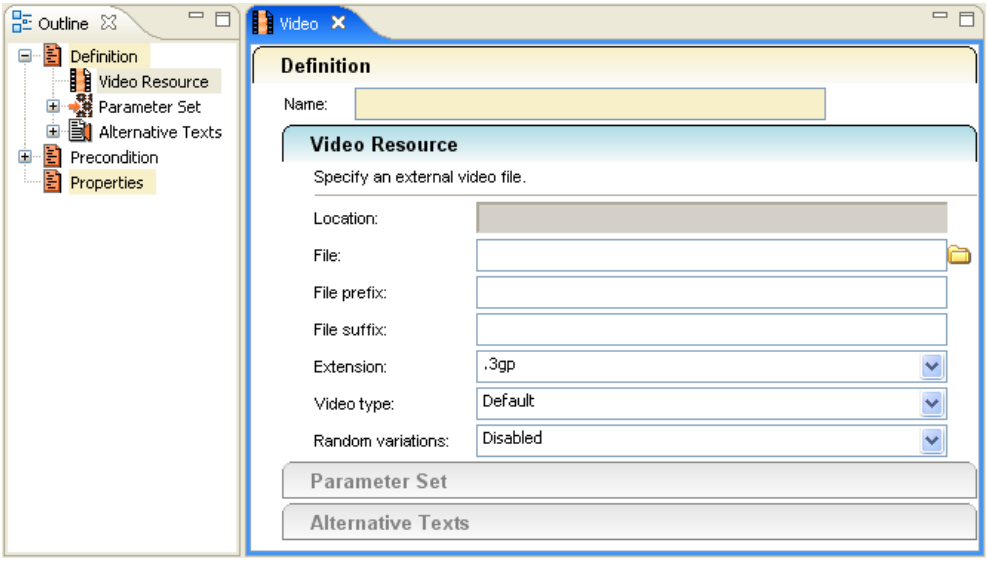
Copyright © 2001-2011 Voxeo Germany GmbH

Note: Some media platforms do not allow to playback audio and video files in the same call and/or to use speech recognition in a video call. Some also restrict the number of video files per Output item to 1. For more information refer to the documentation of your platform.

Object Definition

The **Definition** of the Video object provides the following sections:

- Video Resource
  To specify an external video file.
- Parameter Set
  To specify an optional set of parameters to be passed to video generation engines.
- Alternative Texts
  To specify a text description of the video that will be shown in project documentation and the storyboard.



For information regarding additional object configuration refer to *Precondition* and *Properties* in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Video Resource

The **Video Resource** section provides the following properties:

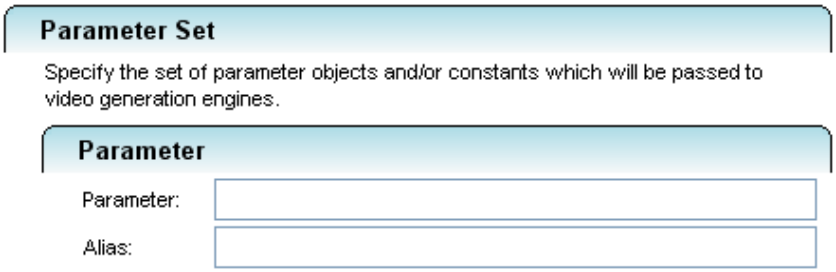
Property	Description
Location	Optional parameter to specify the location (using a Resource Locator object) of the video file to be used.
File	Defines the file name of the video file to be used. The name of the file can either be typed in or selected through a file browser by clicking the **Browse** button to the right of the **File** field. The filename can also be provided dynamically during call time by assigning a Variable, Expression, Script, or Layer object. In Desktop for Web the **Browse** button will only be displayed if the Resource Locator object is available and includes a physical path definition. Moreover the video file can be played by clicking the **Play** button to the right of the **File** field. This will start the default media player associated with Internet Explorer and play the video file in this player. This button will only be displayed if the Resource Locator object is available and includes a URL definition without references to dynamic objects (Variable, Expression, Script, or Layer).
File prefix	Optionally specifies a prefix that is applied to the filename. Instead of providing a static value, the prefix can also be supplied by using a Variable, Expression, Script, or Layer object.
File suffix	Optionally specifies a suffix that is appended to the filename. Instead of providing a static value, the suffix can also be supplied by using a Variable, Expression, Script, or Layer object.
Random variations	Defines whether randomizing is enabled for the video file. To enable randomizing, set the **Random variations** property to a number between 2 and 99, by selecting an appropriate value from the drop-down list. If randomizing is enabled and different video files are provided with the specified filename extended by different numbers (e.g. *teaser1.3gp*, *teaser2.3gp*, *teaser3.3gp*), the server will randomly pick one of the available files with a random file extension between 1 and the selected number whenever the Video object is processed. The server ensures that all available video files are played before a repetition occurs. By default, the **Random variations** property is set to *Disabled* indicating that no randomization is used. For more information on randomization see the Output object in this *Object Reference*.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Video type	Sets the type of the video resource that is defined in the Video object. It can be one of *Default*, *None*, *Video*, and *Video+Audio*. Depending on the kind of content of the video file, you can set this to *Video* (meaning the file does not have any audio content) or *Video+Audio* (meaning the file holds video and audio information). This is a setting necessary for the media platform, which will be streaming the file to the caller's device. If you leave it at *Default*, the default setting from the Service object definition will be used. This makes sense if all of your files are of the same type (typically *Video+Audio*). You can optionally change the default setting at call time using the VIDEOTYPE function through an Expression object. Setting it to *None* means that no information on the content will be passed to the media platform. Instead of selecting one of the provided types from the drop-down list, the Video type can also be supplied by using a Variable, Expression, Script, or Layer object.
Extension	Specifies the video file extension. Instead of selecting one of the provided file extensions from the drop-down list, the extension can also be supplied by using a Variable, Expression, Script, or Layer object. If the filename already contains the extension, set this field to *None*. The complete filename for the video file is generated at call time according to the following formula: `[prefix] + file + [suffix] + [random number] + [extension]`

Parameter Set

The **Parameter Set** section is optional and relevant only for Video objects that refer to external video generation engines. Use it to define any number of parameters, all of which will be added to the final URL as CGI parameters, in order to control the engine if required.



This section provides the following properties:

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Parameter	Specifies the value of the parameter. Possible parameter types are Variable, Collection, Expression, Layer, Resource Locator or Script objects, and constants. When using a Resource Locator object as a parameter, its URL definition is passed to the script. The field can be left empty to denote an empty value.
Alias	Optionally specifies the name of the parameter. If left empty, the reference ID of the object defined in the **Parameter** field will be used as the name. This means in particular that for constant values an alias needs to be defined (as they do not have a reference ID). Constant parameters without a defined alias are ignored.

If there is only one parameter (name + value) defined, it will get appended to the end of the full URL with the following syntax:

```
[Resource Locator] + [prefix] + file + [suffix] + [random number] +
[extension] + "?name=value"
```

Example:

Suppose the file and extension are *teaser.clip*, the parameter name evaluates to *bgimage* and the value evaluates to *background.gif*. The resulting URL might look like this:

```
http://127.0.0.1:8081/VoiceObjects/Resources/teaser.clip?bgimag
e=background.gif
```

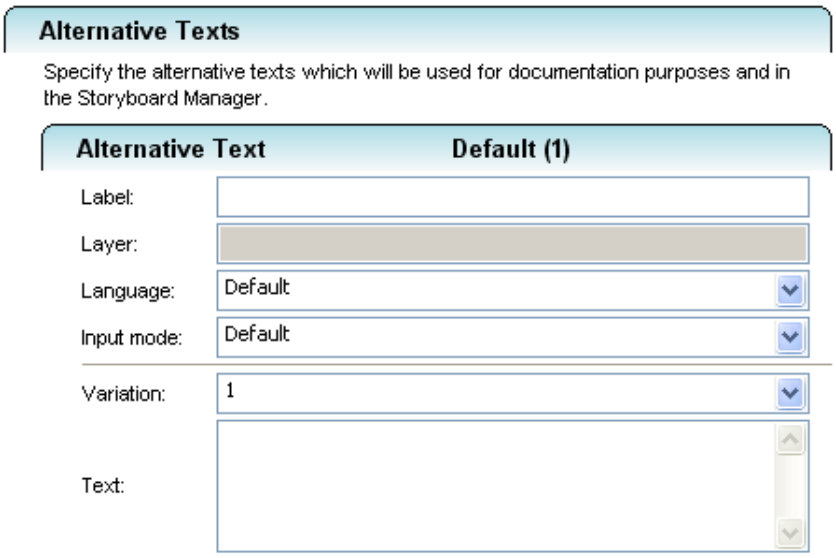
If more than one parameter is defined, they will be added like this:

```
[Resource Locator] + [prefix] + file + [suffix] + [random number] +
[extension] + "?name1=value1&amp;name2=value2&amp;..."
```

Alternative Texts

The **Alternative Texts** section is mainly used to describe the content of the video file(s), typically a script of the audio part of the video. It will appear both in a storyboard export as well as project documentation. By default, it is not rendered into VoiceXML by the server, since none of the media platforms certified with VoiceObjects currently supports alternative text. Nonetheless, the `<renderAltText>` element in the *MPDrivers.xml* file can be set to *true* for a media platform to enable the rendering of alternative text.

Copyright © 2001-2011 Voxeo Germany GmbH



The **Alternative Text** items specify the various texts for video variations, in case the Video object represents more than one file. The text entered into the **Text** field must not contain special characters such as quotation marks, apostrophes, ampersands, angle brackets etc. The alternative text can be typed in or can be set at call time by a Variable, Expression, Script, or Layer object.

In addition to the **Text** field, the following properties can be configured:

Property	Description
Label	Optional parameter to identify the alternative text item in a list.
Layer	Associates the alternative text item with a custom layer. This item is only eligible for playback if the layer evaluates to *true* at call time. If you reference Video objects in different Output items using different layer conditions, there is no need to also define the same layers in the Video object itself. If, on the other hand, different outputs are achieved by using a Layer (or Variable, Expression, Script) object within the resource locator definition of the Video object, you can define different alternative texts by associating items with the corresponding layer condition.
Language	Video objects can represent content in multiple languages through dynamic resource locators. The **Language** drop-down list allows you to specify a language for a video file, so that videos containing different languages can be incorporated into the Video object. When dealing with monolingual applications, this field can be ignored.
Input mode	Associates the alternative text item with a specific input mode. Leave this field at the default value if your application does not distinguish between voice and DTMF input in the wording of your application prompts. Otherwise, tag the item with the appropriate mode.

Copyright © 2001-2011 Voxeo Germany GmbH

Property	Description
Variation	If randomizing is enabled, this field allows you to associate a text with a specific video variation number. If the Video object only refers to a single file, this can be left at the default value. Do not define multiple alternative text items with the same variation number (and the other settings being identical), as this may lead to unexpected behavior at call time. If you want to use the same alternative text for all randomized video files, you only need to define an alternative text for variation number 1.

VoiceObjectsXML Definition

The Video object is represented by the VoiceObjectsXML element `<video>`. It has seven attributes and one group of children.

In addition, the element has the standard attributes described in the *XDK Guide*.

The `<video>` element uses the embedded `<altText>` element.

Video

Attributes

- **`location`**
  Defines the location from where the video file is to be retrieved. Must be a reference to a Resource Locator object.
- **`file`**
  The file name of the video file, without extension. Can be a constant name, or a reference to a Variable, Expression, or Script object.
- **`prefix`**
  A prefix to be used together with the video file name.
- **`suffix`**
  A suffix to be used together with the video file name.
- **`random`**
  Indicates whether randomization should be used. Can be either *disabled* or an integer >= 2 specifying the number of variations in which the file is available. If not specified, defaults to *disabled*.
  For details see Random variations above.
- **`videoType`**
  Specifies the type of video data to be played, which influences the MIME type with which the file is tagged. One of *default*, *video*, *videoaudio*, *none*. If not specified, defaults to *default*, which indicates that the default video type defined in the service is used.
- **`videoFileExtension`**
  Specifies the file extension for the video file as either a constant value or a reference to a Variable, Expression, or Script object. Legal constant values are *none, 3g2, 3gp, avi, bmp, clip, gif, jpeg, jpg, m4v, mp4, mp4v, rtp*. If not specified, defaults to *3gp*.

Copyright © 2001-2011 Voxeo Germany GmbH

The complete filename for the video file is generated at call time according to the following formula:

```
[prefix] + file + [suffix] + [random number] + [extension]
```

Children

- **<expression usage="precondition">** or **<variable usage="precondition">** or **<collection usage="precondition">** or **<script usage="precondition">**
  Defines the precondition for the Video object.
- **?<parameterSet>**
  Defines the parameter set.
- ***<altText>**
  Defines the corresponding alternative text(s).

Example

```
<video location="#Video locator" file="welcome"
videoFileExtension="3gp"/>

<video location="#Video locator" file="trailer"
videoFileExtension="mp4" random="5"/>
```

ParameterSet

Children

- **+<item>**
  Defines the list of entries in the parameter set. The use of the `alias` attribute is optional when the `object` attribute is used, and required when the `value` attribute is used.

Example

```
<parameterSet>
  <item alias="bgimage" object="#BGImage"/>
  <item alias="bgcolor" value="blue"/>
</parameterSet>
```

Item

Attributes

- **object**
  Defines a reference to an object. If this attribute is defined, the `value` attribute must not be defined.
- **value**
  Defines a constant value. If this attribute is defined, the `object` attribute must not be defined.
- **alias**
  Defines the parameter alias.

Copyright © 2001-2011 Voxeo Germany GmbH

Example

```
<item alias=”bgcolor” value=”blue”/>
```

AltText

The <altText> element defines the alternative texts to be used for the Video object. It may only occur embedded inside an <video> element.

The <altText> element has six attributes and contains text as a child.

Attributes

- **label**
  A text string providing a name for the alternative text.
- **random**
  Defines the variation (random index) to which this alternative text belongs. An integer >= 1. If not specified, defaults to 1.
- **language**
  Defines the language for which this alternative text is valid. Can be *default* or a valid language code (e.g. *de-DE*, *en-US*, etc.). If not specified, defaults to *default*. Appendix A – Language Codes contains a list of all language codes available in VoiceObjects together with the respective language they represent.
- **inputMode**
  Defines the input mode for which this alternative text is valid. Can be *default*, *voice*, *dtmf*, or *voicedtmf*. If not specified, defaults to *default*.
- **layer**
  Defines the layer for the alternative text. Can either be a reference to a Collection, Expression, Script, or Variable object; or a layer state reference of the form “Layer=State” or “Layer!=State” where “State” is the label of a state for the layer “Layer”.
- **object**
  Optional reference to an Expression, Variable, or Script object that defines the alternative text.

Children

- **CDATA**
  The static alternative text itself. Note that the text may not contain special characters such as apostrophes, quotation marks, angle brackets etc.

Example

```
<altText language=”de-DE”>
  Bitte geben Sie Ihre Telefonnummer ein.
</altText>

<altText random=”1”> Sorry? </altText>
<altText random=”2”> Once more, please. </altText>
<altText random=”3”> I didn’t quite get that. </altText>
```

Copyright © 2001-2011 Voxeo Germany GmbH

Object Interoperability

The following table contains all object types that can reference a Video object:

Icon	Object Name	Use Case Example
	Output	A Video object can be referenced within an autonomous or embedded Output object definition.
	List	A Video object can be referenced within the content formatting definition of a List object, to play back a video file as part of the List content.

Object Naming Conventions

In order to leverage the capabilities of the integrated documentation of VoiceObjects it is important to provide intuitive and self-explanatory object names and descriptions.

The name of a Video object should provide information about what is being played. The table below lists two examples:

Name	Description
 Trailer	Plays a trailer for the motion picture.
 Tutorial	Provides the caller with a tutorial on how to use the service.

Copyright © 2001-2011 Voxeo Germany GmbH

A - Language Codes

The following list shows all language codes available in VoiceObjects together with the language they represent.

Internal Code	Language (English)	Language (German)
Default	Default	Standard
sq-AL	Albanian	Albanisch
ar-	Arabic	Arabisch
ar-AE	Arabic (U.A.E.)	Arabisch (V.A.E.)
ar-JO	Arabic (JO)	Arabisch (JO)
bg-BG	Bulgarian	Bulgarisch
ca-ES	Catalan (ES)	Katalanisch
cn-HK	Cantonese	Kantonesisch
cs-CZ	Czech	Tschechisch
dk-DK	Danish	Dänisch
nl-	Dutch	Niederländisch
nl-BE	Dutch (BE)	Niederländisch (BE)
nl-NL	Dutch (NL)	Niederländisch (NL)
en-	English	Englisch
en-AU-NZ	English (AU-NZ)	Englisch (AU-NZ)
en-IN	English (IN)	Englisch (IN)
en-SG	English (SG)	Englisch (SG)
en-UK	English (UK)	Englisch (UK)
en-US	English (US)	Englisch (US)
en-ZA	English (ZA)	Englisch (ZA)
eu-ES	Euskara (ES)	Baskisch (ES)
fil-PH	Filipino	Filipino
fi-FI	Finnish	Finnisch

Copyright © 2001-2011 Voxeo Germany GmbH

Internal Code	Language (English)	Language (German)
fr-	French	Französisch
fr-BE	French (BE)	Französisch (BE)
fr-CA	French (CA)	Französisch (CA)
fr-CH	French (CH)	Französisch (CH)
fr-FR	French (FR)	Französisch (FR)
gl-ES	Galician (ES)	Galizisch (ES)
de-	German	Deutsch
de-AT	German (AT)	Deutsch (AT)
de-CH	German (CH)	Deutsch (CH)
de-DE	German (DE)	Deutsch (DE)
el-GR	Greek	Griechisch
he-IL	Hebrew	Hebräisch
hi-IN	Hindi	Hindi
hu-HU	Hungarian	Ungarisch
it-	Italian	Italienisch
it-CH	Italian (CH)	Italienisch (CH)
it-IT	Italian (IT)	Italienisch (IT)
jp-JP	Japanese	Japanisch
ko-KR	Korean	Koreanisch
mk-MK	Macedonian	Makedonisch
ms-MY	Malay	Malaysisch
zh-	Mandarin	Mandarin
zh-CN	Mandarin (CN)	Mandarin (CN)
zh-TW	Mandarin (TW)	Mandarin (TW)
no-NO	Norwegian	Norwegisch
pl-PL	Polish	Polnisch

Copyright © 2001-2011 Voxeo Germany GmbH

Internal Code	Language (English)	Language (German)
pt-	Portuguese	Portugiesisch
pt-BR	Portuguese (BR)	Portugiesisch (BR)
pt-PT	Portuguese (PT)	Portugiesisch (PT)
ro-RO	Romanian	Rumänisch
ru-RU	Russian	Russisch
es-	Spanish	Spanisch
es-AR	Spanish (AR)	Spanisch (AR)
es-CL	Spanish (CL)	Spanisch (CL)
es-CO	Spanish (CO)	Spanisch (CO)
es-ES	Spanish (ES)	Spanisch (ES)
es-LA	Spanish (LA)	Spanisch (LA)
es-MX	Spanish (MX)	Spanisch (MX)
es-PE	Spanish (PE)	Spanisch (PE)
es-US	Spanish (US)	Spanisch (US)
sl-SI	Slovenian	Slowenisch
sk-SK	Slovak	Slowakisch

Copyright © 2001-2011 Voxeo Germany GmbH