

Using Dynamic Grammars with VoiceObjects

Integration with the NuGram Server



VoiceObjects

Nov 26, 2008

Dr. Andreas Volmer (avolmer@voiceobjects.com)



Using Dynamic Grammars with VoiceObjects

VoiceObjects 7.3/7.4

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 © 2001-2009 VoiceObjects Germany GmbH and its licensors. All rights reserved.

Published in Germany – Legal information January 2009

Information in this document is subject to change without notice and does not represent a commitment on the part of VoiceObjects or any of its subsidiaries. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. You may not copy, use, modify, or distribute the software except as specifically allowed in the license or nondisclosure agreement. 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 VoiceObjects or a subsidiary thereof.

Protected by German and European 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 VoiceObjects by e-mail at vo-documentation@voiceobjects.com.

Trademarks

VoiceObjects is a registered trademark of VoiceObjects Germany GmbH. 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. VoiceObjects is not making any representation or granting any warranty with respect to such material, and the distribution of such material shall not subject VoiceObjects to any liability.

Explicit Copyright Notice

The VoiceObjects software includes software developed by the Apache Software Foundation (www.apache.org). Copyright © 1999-2009 – 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 contain copyright material authorized to copy from Bocaloco Software LLC (www.xmlbuddy.com). All rights to such copyright material rest with Bocaloco Software.

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.

Document Number: 036-20081126-VO7



Table of Contents

TABLE OF CONTENTS	3
1 OVERVIEW	4
Use Cases for Dynamic Grammars	4
The NuGram Server Approach	5
Implementation Overview	5
2 IMPLEMENTATION DETAILS	7
The Grammar Server Connector	7
Connector item configuration	7
Parameter Set	7
NuGram Server configuration	8
Managing Context Information	9
Sample context collection	10
Manipulating collections	12
Releasing the Session on the Grammar Server	12
3 SAMPLE APPLICATION	13
Installation and Configuration	14
Configuration Alternatives	14



1 Overview

The VoiceObjects Grammar Server connector integrates NuEcho's NuGram dynamic Grammar Server with the VoiceObjects platform.

It allows VoiceObjects developers to invoke the NuGram Server from VoiceObjects applications with minimal development effort. For the technical integration, it uses the NuGram Server HTTP API and VoiceObjects Java connector.

Use Cases for Dynamic Grammars

There are many situations when dynamic grammars can or should be used in phone self services. Here are a few example use cases¹:

- **Address capture**
In order to capture the address of a caller, an application might first ask for the caller's postal or zip code and then ask for the address using an address recognition grammar dynamically built based on a list of address records associated to the recognized postal or zip code.
- **Voice dialing**
A voice dialing application could use a recognition grammar dynamically generated from the data in a user's address book. The grammar could support sentences such as "Call John Smith", "John Smith at home", "Call John Smith's cellular", etc.
- **Personalized bill payee list**
In a banking bill payment application, the payee list grammar is dynamically generated based on the list of payees that has been set up by the user.
- **Identity validation**
Many applications use security questions to validate the identity of the caller. Based on an identity claim (e.g., a social security number or a telephone number), the application asks the caller to answer security questions based on information contained in the caller's profile, for instance a mother's maiden name, the name of a pet, a secret word, etc. In this case, because the range of possible responses would often be too large, some of the recognition grammars need to be dynamically built based on the expected responses.
- **One-step correction**
Let's suppose an address recognition N-best list contains the following hypotheses: "four fifty main street", "four sixty main street", and "four fifty-one main street" and let's suppose the caller has actually spoken the third hypothesis. Suppose also that, when confirming the first hypothesis to the caller, we use a confirmation grammar that covers corrections that the caller is likely to make when being proposed an incorrect choice (e.g., "no, four sixty-one"). In other words, the confirmation grammar is built based on hypotheses found in the recognition result. This would make it possible to recognize the eventual correction and act on it, thereby avoiding unnecessary interactions with the caller and, as a result, contributing to enhanced user experience and success rate.

¹ From: „Use cases for dynamic grammars“ by Yves Normandin, <http://blog.nuecho.com/2008/10/06/use-cases-for-dynamic-grammars/>. This document discusses more use cases and technical reasons for dynamic grammars.



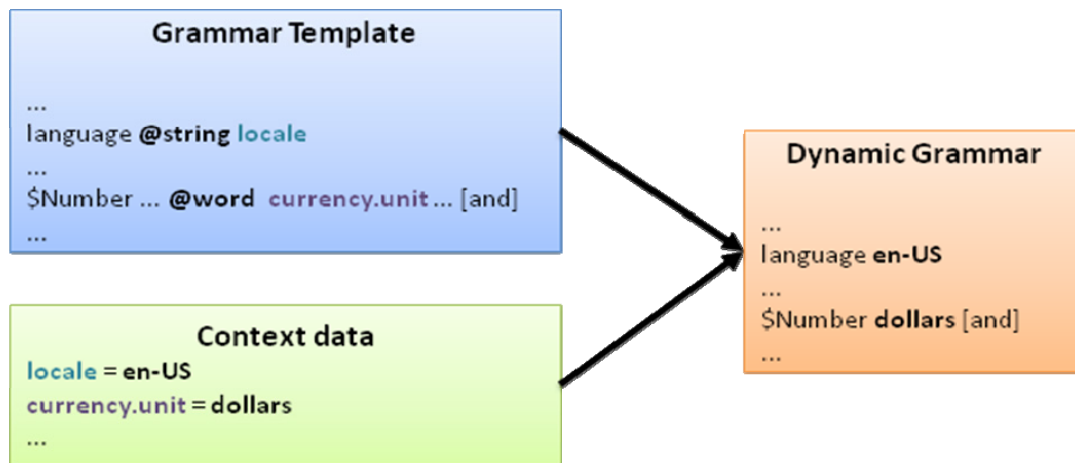
The NuGram Server Approach

NuGram Server is an HTTP-based server that offers grammar-related services, including dynamic grammar instantiation, source grammar generation to various formats (ABNF, GrXML, GSL), and sentence interpretation.

Dynamic grammars can be created at call time, based on grammar templates that are dynamically instantiated using customer and application specific context data. The following sketch illustrates this principle:

A **grammar template** is defined at *design time*, leaving placeholders to be filled in dynamically. NuGram server supports a rich Grammar Language² for creating dynamic grammars.

At *call time*, the application specifies **context data**, which is then used to instantiate a **dynamic grammar**, mapping placeholders to actual data.



Implementation Overview

From a bird's eye view, using the NuEcho Grammar Server from a VoiceObjects application involves the following resources and dialog flow objects:

- A grammar template (which can be created with the NuGram Grammar IDE, using the tags defined in NuEcho's Grammar Language);
- a Collection object that holds the data used for instantiating the dynamic grammar;
- a Connector object that uses the VoiceObjects Grammar Server connector;
- an Input object that references the dynamically instantiated grammar.

Let's have a closer look at the VoiceObjects Grammar Server connector (step 3 above).

When this Connector object is executed, it performs the following steps, communicating with the NuGram Server via HTTP³:

² https://www.grammarserver.com/html_app/doc/abnf_tutorial.

³ https://www.grammarserver.com/html_app/doc/grammar_server



1. The grammar template is uploaded to the Grammar Server if required (i.e. only once per lifetime of a VoiceObjects Server instance).
2. Context information from the VoiceObjects Collection object is transformed into JSON format, as specified by the NuGram Server HTTP API.
3. The dynamic grammar is instantiated, using the context information from the previous step.
4. The return value can be either a URL to the grammar file thus instantiated, or the full content of the grammar file. The grammar format can be any format supported by NuGram Server and may be different from the format of the original grammar template.



2 Implementation Details

The Grammar Server Connector

The VoiceObjects Grammar Server connector is implemented as a Java connector. The class that is instantiated from VoiceObjects is *com.voiceobjects.nugram.GrammarServerConnector*, which is deployed in the Java archive *GrammarServerConnector.jar*. The source code is available with this package, along with some extra Java archives that are not already contained in the standard VoiceObjects lib folder.

The following two tables give details on the parameterization of the Connector item. First, the configuration fields of the Connector item are specified, and second, the parameter set. Note that the parameter set only uses *request* parameters; the (single) *return* value is assigned to the variable that is linked into the **Return value** field.

Connector item configuration

Field	Content	Description
Location	<i>Connector Locator</i> (Resource Locator object)	Resource locator with a URL for the Java archives referenced in the File field.
File	GrammarServerConnector.jar, commons-collections-3.2.1.jar, ezmorph-1.0.5.jar, json-lib-2.2.2- jdk15.jar	Java archives containing the compiled connector code and referenced libraries.
Class/Port	com.voiceobjects.nugram. GrammarServerConnector	Class name of the Java connector.
Method	getGrammar	This method implements the instantiation and (optional) fetching of the dynamic grammar.
Return value	<i>Dynamic Grammar</i> (Variable object)	This variable will hold the result of the connector execution: Either the URL of the dynamic grammar or the content of that grammar, depending on whether or not the parameter <i>fetchGrammarContentWithExtension</i> was provided.

Parameter Set

Alias / Property	Parameter	Description
sessionID	<i>System:Dialog ID</i> (Expression object)	The VoiceObjects dialog ID is used as session ID on the Grammar Server.
grammarTemplateFile		Name of the grammar template, e.g. "Currency.abnf"



Alias / Property	Parameter	Description
grammarTemplatePath	<i>Grammar Locator</i> (Resource Locator object)	Resource locator pointing to the URL of the grammar template(s).
contextFromCollection	Grammar Context (in VoiceObjects Collection format)	The Collection object holding the dynamic data that is used as context data for the instantiation of the dynamic grammar. For more details, see below. When providing this parameter, do not provide the <i>context</i> parameter.
context	Grammar Context (string in JSON format)	As an alternative to using a Collection object for holding the dynamic data, you can provide a string in this parameter. This string must contain the data in JSON format and will be passed on to the Grammar Server as-is, without further processing. When providing this parameter, do not provide the <i>contextFromCollection</i> parameter.
fetchGrammarContent WithExtension		If this parameter is left empty or is not provided as a parameter at all, the return value will be populated with the full URL of the dynamic grammar, without the extension. This URL can then be used in the File field of a Grammar item. In this case, the Grammar item's Extension field must be properly selected. If a grammar extension is provided, e.g. <i>grxml</i> or <i>abnf</i> , the return value will be populated with the grammar content in the desired format. This value can then be used as-is in the Grammar field of a Grammar item (with the Text-To-Grammar field clear).
configuration	<i>Grammar Server Configuration</i> (Variable object)	Configuration parameters for accessing the Grammar Server, such as user name and password, proxy settings, encoding, and more. For more details, see below.

NuGram Server configuration

The configuration variable that is used with the alias *configuration* in the connector's parameter set must contain a comma-separated list of configuration settings.

Example content:

proxy.server=172.50.16.198, proxy.port=80, user.name=av, user.password=test

The following settings can be configured. If no value is provided, the default value is assumed. Note that the proxy settings are only necessary if your VoiceObjects Server accesses the internet through a Web proxy.



Key	Default	Description
<i>server.host</i>	www.grammarserver.com	Hostname of the NuGram Server
<i>server.port</i>	8082	Port of the NuGram Server
<i>user.name</i>		Login name of the NuEcho developer account
<i>user.password</i>		Login password of the NuEcho developer account
<i>charset</i>	UTF-8	Character set of the dynamic grammar, e.g. UTF-8 or ISO-8859-1
<i>proxy.server</i>		Hostname of HTTP Proxy server required to access the NuGram Server [optional]
<i>proxy.port</i>		Port of HTTP Proxy server required to access the NuGram Server [optional]

Managing Context Information

The context information is used for configuring dynamic grammars with actual, session- and customer-specific values. Data from the context replaces placeholder tags in grammar templates when a grammar is instantiated.

The NuGram Server HTTP API requires the context information to be provided in JSON format. This format conversion is automatically performed by the connector which takes a VoiceObjects collection as input and converts it to JSON formatted string.

Dynamic elements in grammars can be defined in a number of ways, using NuEcho's Grammar Language syntax. Examples for such dynamic grammar directives embedded in ABNF grammars are:

```
language @string language;  
...  
@word user.id  
...  
@alt  
  @for (unit : currency.units)  
    @word unit  
  @end  
@end  
...
```

A collection providing information for these placeholders could look like this:

```
<root>  
  <row>  
    <col name="key">language</col>  
    <col name="value">en-UK</col>  
  </row>  
  <row>  
    <col name="key">user.id</col>  
    <col name="value">5904</col>  
  </row>  
</root>
```



```

    <col name="type">number</col>
  </row>
  <row>
    <col name="key">currency.units</col>
    <col name="value">dollar|dollars|bucks</col>
    <col name="type">list</col>
  </row>
</root>

```

Behind the scenes, this collection will be transformed to the following JSON string:

```

{"language": "en-UK", "user": {"id": "5904"},
 "currency": {"units": ["dollar", "dollars", "bucks"]}}

```

Note that the *keys* exactly match the names of the placeholders in the grammar template.

Each row in the collection must provide the following columns:

- A *key* column, matching the name of an expression of a `@string`, `@word`, or `@tag` directive in the grammar template;
- A *value* column with the content that will replace that expression.

A third column with name="type" is optional. It can have the following values:

Type	Sample	Description
string (default)	dollar	The value is interpreted as string content.
boolean	True	The value must be <i>true</i> or <i>false</i> , and it can be used in <code>@if</code> directives in the grammar template.
number	453.54	In expressions that go with the <code>@tag</code> directive, it can be important to distinguish alphanumeric from numeric content.
list	dollar dollars bucks	A list of utterances, separated by the pipe symbol. This can be used in <code>@for</code> directives.
array		This type is special. It is used as a marker indicating that the following rows all belong to one value instance in an array. See example below.

Sample context collection

```

<root>
  <row>
    <col name="key">contacts</col>
    <col name="value">1</col>
    <col name="type">array</col></row>
  <row>
    <col name="key">contacts.id</col>
    <col name="value">15</col></row>
  <row>
    <col name="key">contacts.names</col>
    <col name="value">John Smith|Jonnyboy</col>
    <col name="type">list</col></row>
  <row>
    <col name="key">contacts.hasPhoneNumber</col>
    <col name="value">>true</col>

```



```
        <col name="type">boolean</col>
</row>
<row>
        <col name="key">contacts.address</col>
        <col name="value">460 albert street montreal h3b1a7</col>
</row>
<row>
        <col name="key">contacts.phone</col>
        <col name="value">4960104985</col>
        <col name="type">number</col>
</row>
<row>
        <col name="key">contacts.email</col>
        <col name="value">john.smith@nuecho.com</col>
</row>
<row>
        <col name="key">contacts</col>
        <col name="value">2</col>
        <col name="type">array</col></row>
<row>
        <col name="key">contacts.id</col>
        <col name="value">254</col></row>
<row>
        <col name="key">contacts.names</col>
        <col name="value">Michael Johnson|Mike</col>
        <col name="type">list</col></row>
<row>
        <col name="key">contacts.hasPhoneNumber</col>
        <col name="value">>false</col>
        <col name="type">boolean</col>
</row>
<row>
        <col name="key">contacts.email</col>
        <col name="value">michael.johnson@nuecho.com</col>
</row>
</root>
```

Note that there are two sets of “contact” data, each initiated with a “type=array” row in the Collection object.

Below is the result of the internal conversion to JSON executed by the connector. Note that the line breaks have been entered for ease of reading; the JSON string that is actually generated contains a single line.

```
{
  "contacts":
  [
    { "id":"15",
      "names":["John Smith","Jonnyboy"],
      "hasPhoneNumber":true,
      "address":"460 albert street montreal h3b1a7",
      "phone":4960104985,
      "email":john.smith@nuecho.com
    },
    { "id":"254",
      "names":["Michael Johnson","Mike"],
      "hasPhoneNumber":false,
      "email":michael.johnson@nuecho.com
    }
  ]
}
```



Manipulating collections

In VoiceObjects, collections can be manipulated with a number of dedicated expression functions. For the purpose of setting dynamic values into the context collection, the most important functions are

FINDROW (collection, colName, searchValue, [startRow], [caseSensitive])

ADDROW (destCollection, srcCollection, srcRow)

:= (collection, rowIdx, columnIdx, arg)

For further information, refer to the Expression object in the *Object Reference* in the VoiceObjects product documentation.

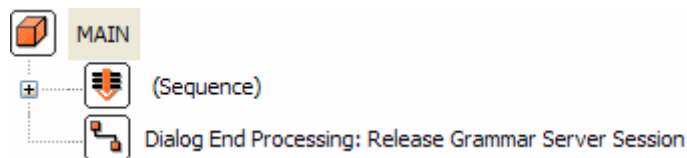
And in order to see how to use these functions in particular to modify data for the dynamic grammar generation, inspect the sample application.

Releasing the Session on the Grammar Server

When a dynamic grammar is instantiated, a session is automatically created on the Grammar Server. We are using the VoiceObjects dialog ID to uniquely identify this session.

When the VoiceObjects session expires – i.e., the call is terminated – it is good practice to release the according session on the Grammar Server. If that session is not deleted, the Grammar Server will do so after a certain timeout; however, for the sake of resource usage it is recommended to explicitly delete the session.

For this reason, the VoiceObjects Grammar Server connector has a second functionality, *deleteSession*, which should be called in the dialog end processing of the main module.



The parameterization of this connector is largely identical to that of the Grammar Server connector as described in [The Grammar Server Connector](#). These are the differences:

- In the Connector item configuration, replace the method name *getGrammar* by *deleteSession*.
- From the parameter set, you only need the *sessionID* and the *configuration* string.



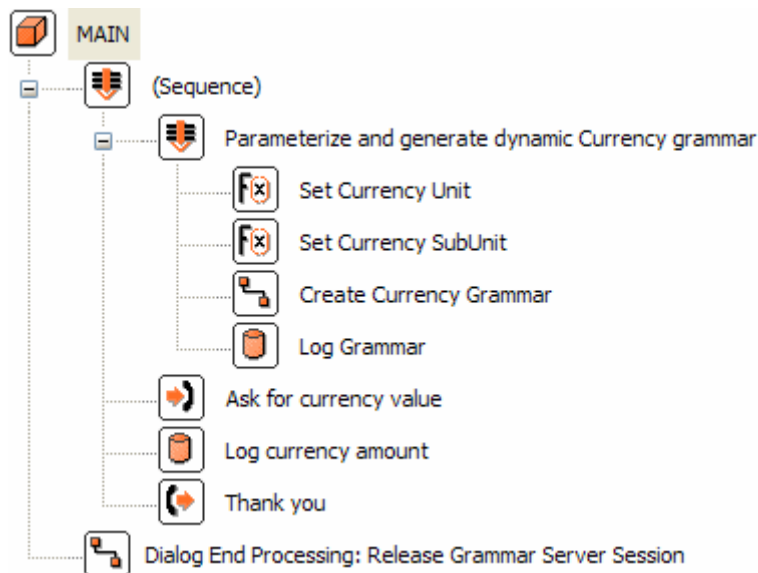
3 Sample Application

The sample application demonstrates how to embed the Grammar Server connector into a dialog flow. It contains a single input state that references a dynamic grammar.

This dynamic grammar is instantiated based on the grammar template *Currency.abnf*. Note that even though the grammar template uses the ABNF format, the dynamic grammars that are generated from it can use any format – including SRGS-XML and Nuance GSL – supported by the Grammar Server.

The sample grammar template represents a currency grammar that allows for utterances like “50 dollars and 30 cents”, “I want to transfer 350 dollars”, or “2 dollars and 50 cents please”. The name of the currency and the name of its sub-unit are treated as parameters: *currency.units* and *currency.subunits*.

For each of these parameters, a list of synonyms can be provided, such as *dollar|dollars* or *pound|pound sterling|british pound* and *cent|cents* or *pence|pennies*, respectively. Also, the grammar language is treated as a dynamic parameter.



Looking at the dialog flow, these are the important steps:

- The Collection object *Currency Grammar Context* is populated with dynamic values (using the **FINDROW** and **:=** functions in the expressions *Set Currency Unit* and *Set Currency SubUnit*).
- The grammar is parameterized, instantiated, and finally fetched from the Grammar Server in the *Create Currency Grammar* Connector object.
- The Input object *Ask for currency value* is executed, using the grammar that was created in the previous step.
- In the dialog end procession of the *MAIN* module, the session on the Grammar Server is deleted.



Installation and Configuration

Copy the contents of the folder *Resources\Samples\GrammarServerDemo* to the *Resources\Samples* folder of your VoiceObjects Server (or Desktop for Eclipse when working in standalone mode) installation.

Create a new project in VoiceObjects Desktop, open the project, and import the project definition file *Grammar_Server_Demo.xml*.

For detailed instructions refer to the *Desktop for Eclipse Guide* or *Desktop for Web Guide* of the VoiceObjects product documentation.

Within the project, adjust the following settings:

- If necessary, adjust the physical path and the URL in the Resource Locator object *Base Locator*.
- Configure the NuGram Server access configuration in the Variable object *Grammar Server Configuration* (for details, refer to the section [NuGram Server configuration](#)). In particular, provide your account name and password from the NuGram developer Website, and specify proxy settings if your VoiceObjects installation accesses the Web through a Web proxy server. If you have direct Internet access, make sure that no *proxy.server* or *proxy.port* settings are specified in the configuration string.

Configuration Alternatives

In the default configuration of this sample application, the connector has the *fetchGrammarContentWithExtension* parameter set to *grxml*. This instructs the connector to fetch the full content of the dynamical grammar from the Grammar Server in order to use it as inline grammar (in the SRGS-XML format) in the Input object *Ask for currency value*.

If you have direct access to the Internet (i.e. not going through a proxy server) you may change the configuration such that the connector only returns a URL to the dynamic grammar, e.g.

```
http://www.grammarserver.com:8082/grammar:avolmer/OVAP1f86174c73a707d68a8e5f5a0000011dbe4790f7/F9BA31017558701FB560004D08196161AF86B065
```

This URL can then be used as grammar file reference in the Input object, and the VoiceXML browser will then fetch the dynamic grammar from the Grammar Server.

To change the configuration accordingly, follow these steps:

1. Delete the *fetchGrammarContentWithExtension* parameter from the connector.
2. Open the first Grammar item in the *Ask for currency value* Input object.
3. Clear the **Grammar** field; populate the **File** field with the Variable object *Dynamic Grammar* variable instead and select the proper **Extension** (e.g. *grxml*).

The screenshots below show the Grammar item configuration before and after the change.

