Composer Help

Working with Grammar Builder

4/2/2025

Working with Grammar Builder

Contents

Grammar Builder provides a solution for supplying simple grammars, without requiring GRXML expertise. This editor provides a hierarchical view of certain grammar concepts in a simple, abstract way. Each level of this tree contains properties which affect all child members. Following is a brief description of the key concepts of the Grammar Builder model as well their relationship with GRXML.

Note: A Grammar built with the Grammar Builder is not a GRXML file.

Grammar

The grammar is the root object of the tree. It serves to provide an implicit description of the intended use of a grammar. For example, a grammar which would be used for a bank customer could be called bankmenuinput. The selection of a grammar name is determined by the file name or the related gbuilder file, and its setting influences the file name(s) of any exported GRXML data.

Within the grammar object are properties for the setting of languages. These languages, or locales, indicate support for a particular language. For each locale that is added to a grammar, a distinct GRXML file will be created specifically to support that language. DTMF, or touch-tone input, is considered a language even though it is not spoken.

Rule

Every grammar must contain at least one rule, but may contain many. Rules provide a grouping for a spoken (or DTMF) items. Continuing the bank customer scenario, we could have rules for yes/no responses, another for menu options and perhaps another for branch cities. Rules are the product that is referenced in a voice application.

At a point in an application where we wish to retrieve the branch city, we must refer to that grammar's rule. If an application designer does not specify a rule and instead only specifies the grammar file, the default rule is used. Additionally, a rule may be hidden from outside applications by declaring it as private. Usually this is for more sophisticated grammar cross-referencing, which is not currently supported in the more elementary Grammar Builder.

Keywords

Within a rule are specific keywords that will be used to add intelligence to an end application. Keywords become the value which can be identified within the application for use in branching or other application constructs. However, this keyword is independent of what may actually be spoken and is instead an internal identifier.

To bridge the gap between what a caller says or presses on their keypad, locale-specific synonyms are defined. Remember that the languages supported are defined at the grammar level. It is at this point that those defined languages come into play. Each keyword will have a list of words (synonyms) which relate to the keyword for a given language.

For example, assuming as part of our yes/no rule, we have a keyword for yes. This keyword could

contain the word yes for English, 1 for DTMF and oui for French. Regardless of which locale ends up being used in the running application, yes (the keyword identifier) will be returned.

Working with Grammars will guide you through the process of creating a simple grammar, using a user color selection problem as the example to model.

Using the Grammar Builder

Let's create a simple grammar for use with a project and the Grammar Menu block. Our example will be a user color selection problem. You will perform the following steps:

1. Create a new grammar builder file with initial settings.
2. Add keywords and synonyms for a rule.
3. Save the grammar builder file.
4. Export the grammar builder file to standard GRXML format.

Composer provides a cheat sheet for building a simple grammar file:

- Select **Help** > **Cheat Sheets** > **Composer** > **Building Voice Applications** > **Creating a simple grammar**.

Creating a New Grammar Builder File With Initial Settings

The first step is to create a new grammar builder file and provide its initial settings. Follow these steps:

1. Select **File** > **New** > **Other**.
2. From the New dialog box, expand the **Composer** folder, then expand the **Grammars** folder.
3. Select Grammar builder file and click **Next** to continue.
4. In the Container field of the wizard dialog box, click **Browse** to select a project-specific folder to contain the new .gbuilder file. Genesys recommends `<voiceprojectname>/Resources/Grammars` for the location.
5. Set the file name to use for this grammar. File names should give an indication of the context this grammar will be used in. For example, type `colors.gbuilder` in the File name field.

Note: The Grammar Menu block does not pick up changes automatically if you change your Gbuilder file. To synchronize the block with the latest changes, click on the **Gbuilder File** property of the Grammar Menu block. In the popup make sure that the correct **Gbuilder file** and **RuleID** are selected. Click **OK** to close the dialog. Your diagram will now reflect any menu options changes made in the Gbuilder file.

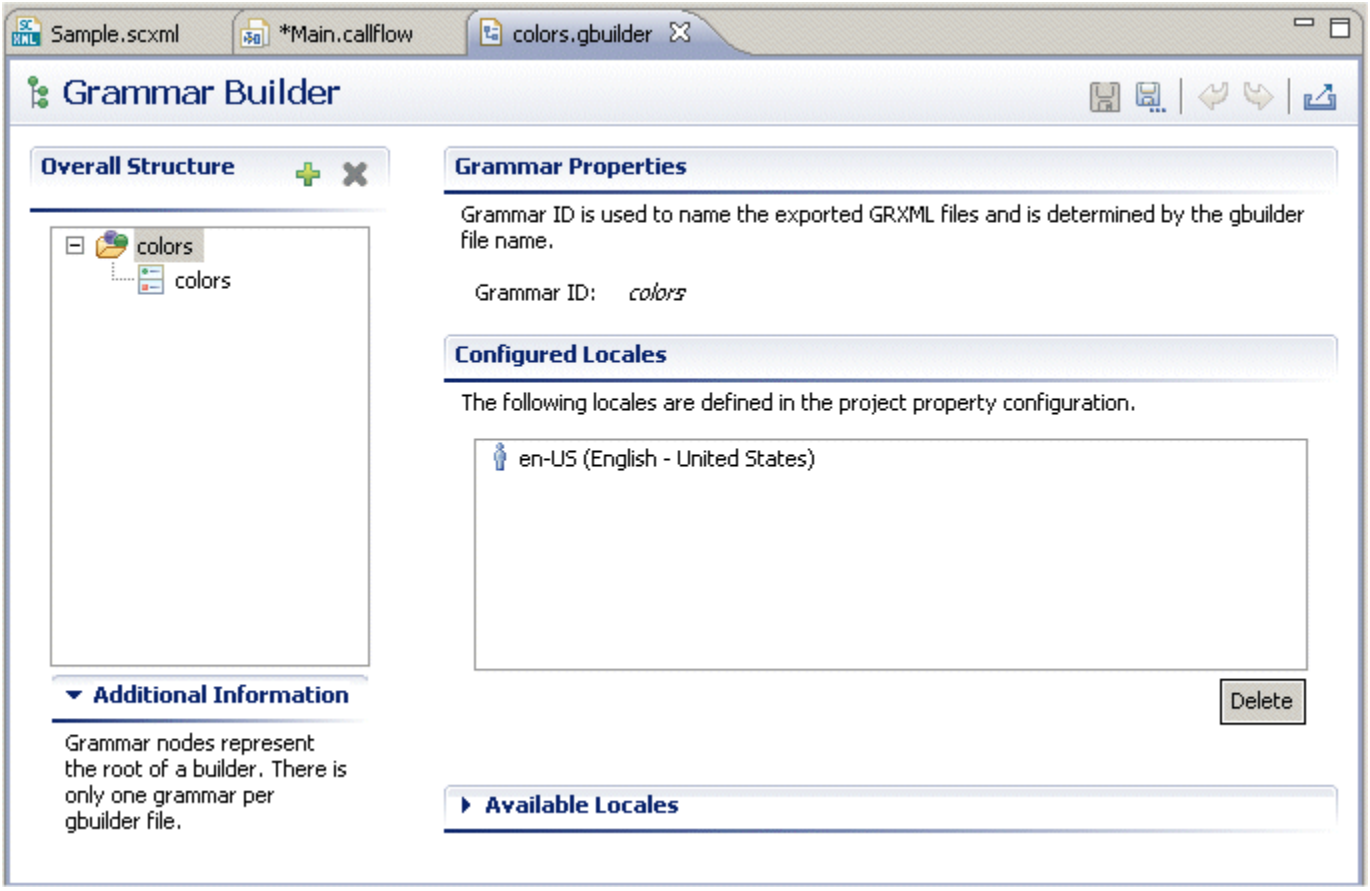
6. Next, set the initial default rule. Rules contain items which form a category. All grammar builder files must have at least one rule. Since our example grammar only deals with one such categorization, type `Colors` in the **Initial Default Rule** field.

7. *Locales* are languages that this grammar will support. By default, **English** and **digit input (DTMF)** are selected in the Initial locale(s) field. If you knew you would need to support additional languages for the grammar, you would select the appropriate check box(es). For our example, the default selections are adequate.

Note: Grammar Builder treats DTMF as a separate language (locale), even though technically it is not categorized as such.

8. After making the selections described above, click **Finish**.

The file is added to the selected project (as you can see in the Project Explorer), and the Grammar Builder opens as shown in the image below.



Adding Keywords and Synonyms for a Rule

Grammar builder files are created with a default rule. The next step is to define keywords for this rule. Each rule can have any number of input-agnostic keywords. These keywords will be returned from either the speech or digit processor for use in your callflow.

By default, a keyword is not usable in an application. This is because multiple languages may use different words/sounds for your keyword. In our example, red may be an appropriate English

pronunciation, but in Spanish this would not be true. Because of this, each configured locale must provide accepted input for the keyword. These inputs are called synonyms. Therefore, keywords consist of a logical identifier and a list of locale-specific synonyms.

Once you have defined keywords and synonyms for the default rule, you can then create additional rules and define keywords and synonyms for those rules as well.

To add a new keyword:

1. Select the **Colors** (default) rule in the **Overall Structure** tree.

    - The Rule Properties area shows the Public Visibility and Default Rule settings for the selected Rule ID.
    - **Default Rule**. This is selected only for the rule that has been set as the default (as is the Colors rule in this example).
    - **Note:** Not all aspects of Composer allow for specific rule targeting within grammar files (grxml). As such, it is highly recommended that you specify a default rule. This rule will be used by default when a reference to the grammar exists that does not target a specific rule. Considering that a default rule (e.g., root) is not mandatory in GRXML, no warning is given when one is not specified
    - **Public Visibility**. If selected, this indicates that this rule can be referenced by an external grammar (in a ruleref element in the grammar making the reference). A public rule can always be activated for recognition. If not selected, the rule is private, which indicates that the rule is visible only within its containing grammar. A private rule can be referenced by an external grammar if the rule is declared as the root rule of its containing grammar.

2. Click the  (**Add**) button.
3. In the Add new keyword dialog box, type a name for this keyword, which is normally an instance of the category that the rule defines. In our example, type **Red** in the **Keyword ID** field and click **OK**.
4. You can repeat the steps above to add more keywords to this rule.

To add a new synonym:

1. Select a keyword from the **Overall Structure** tree. In our example, select **Red**.

The Synonyms area allows you to add synonyms for each of the locales you have defined (each locale is a tab at the bottom of the synonyms table). Note in our example that the window has both English - United States and DTMF as bottom tabs. This allows you to switch the synonym context for the selected keyword.

1. With the English - United States tab selected, click **Add ID** as **Synonym**. This button allows you to add a synonym that is identical to the keyword, thus allowing red to be spoken in English and associated with the keyword **Red**.
2. You may at this time add other values, such as **Crimson** for example, which will also be accepted as Red.
3. Select the **DTMF** tab. To associate the digit 1 with the keyword Red, type 1 in the Digits field and click the Add button.
4. You can repeat the steps above to add more synonyms to this keyword.

Note: If you are using locales representing other languages, the synonyms you create for each locale would represent acceptable values for the keyword in that language. In our example, if you also defined Spanish and French locales, you could create a synonym rojo for the Red keyword in the Spanish locale, and a synonym rouge for the Red keyword in the French locale.

Saving the Grammar Builder File

When you have finished building your grammar builder file, or periodically during the course of building the file, be sure to save the changes you make to the file.

5. To save the file, click  (Save), or to save the file under a different name, click  (Save As) and provide a new file name and location.

Exporting the Grammar to GRXML Format

Because the Grammar Builder saves your grammar to a non-standard GRXML format (denoted with a .gbuilder extension on the file name), you will want to export the grammar to the standard GXML format as follows:

1. Click  (**Export)** , located at the top-right corner of the Grammar Builder editor.
2. If prompted to save, click **Yes**.

You will see a message indicating the file has been successfully exported. The exported GRXML file names are displayed in the success window, and the .grxml file will display in the appropriate locale folder(s) in the Project Explorer under `<voiceprojectname>/Resources/Grammars`. It's important to note that DTMF is considered a locale for the purpose of exportation. As such, an export result for a GBuilder resource with English and DTMF would be placed in `<voiceprojectname>/Resources/Grammars/en-US` and `<voiceprojectname>/Resources/Grammars/DTMF directories`, respectively. These files can now be edited in the GRXML Editor.

Locales and Grammar Builder

When using the Grammar Builder, you specify *locales*, which are the languages that a grammar file will support. The Grammar builder wizard uses the active locales for the Composer Project.

See Locales in CommonBlocks & Functionality.

Dynamic Grammars

Dynamic grammars are used for automated speech recognition (ASR). They are generated "on-the-fly" based on information dynamically pulled out from data sources such as databases, web services,

or the file system. Contrast this to using a static grammar file whose content is fixed. The ASR engine matches the user utterance with the grammar. Returned values are then passed back to the application based on any matches in the grammar.

There are several ways to include dynamic grammars in voice dialogs:

- Use a dynamic VXML page template that creates the dynamic grammar and insert it in-line into the VXML page. Using a dynamic VXML page will provide flexibility in terms of the data source used to generate the grammar.

- If data is being retrieved from a database, using the DB Input block may be another alternative. It generates a grammar based on data retrieved from a database using the DB Data block. It can also generate a grammar based on contents of a JSON array that may have been retrieved from alternate data sources e.g., a Web Service.