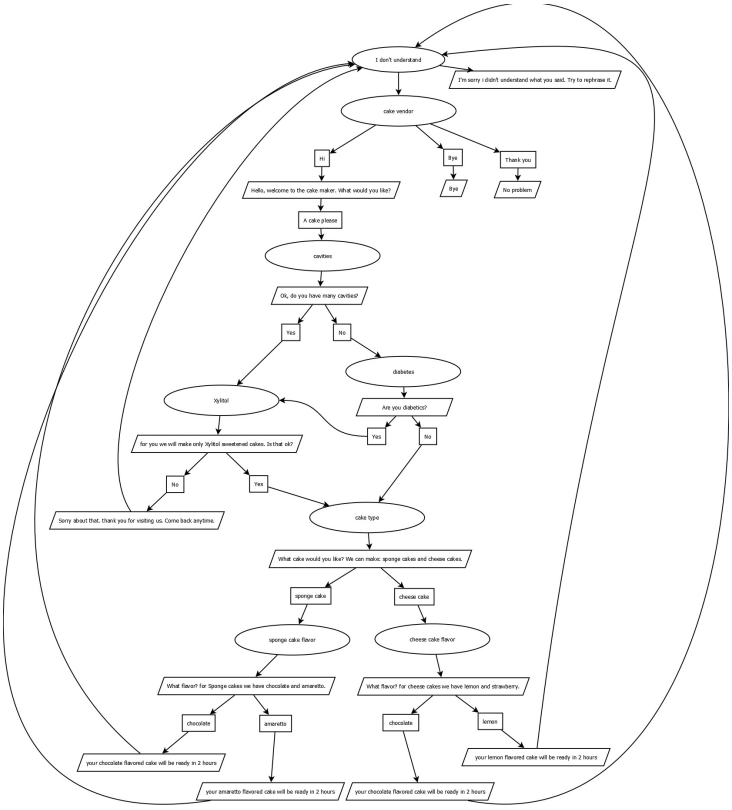
Creating a New Virtual Human with the NPCEditor

Overview

This tutorial will teach you how to create a new character using NPCEditor. The character created in this example will greet the player and then prepare them a cake using a series of yes or no questions. This is the tutorial character's dialog graph:



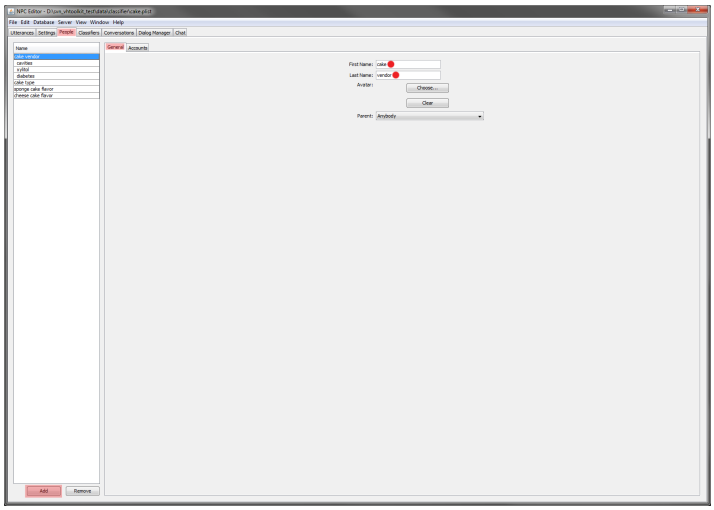
NPCEditor stores data in the p-list format, similar to XML. These files contain the questions and answers for the character, as well as the links between them. There is also an accounts plist required to connect a character to other modules, which is stored in `%AppData%\Roaming\NPCEditor\people\`

Preparation

1. Get a copy of NPCEditor. You can check out a binary out from SVN, or even simpler, grab the latest version of the Virtual Human Toolkit. The Toolkit includes NPCEditor, along with other tools to generate your own virtual human.
2. To run NPCEditor, find the `run.bat` in your NPCEditor install, or from the Toolkit, run the Virtual Human Toolkit Launcher and expand the '<< `Advanced`' options, then click '`Launch`' in the NPCEditor row in the '`Agents`' section.
3. Before you begin, it may be worth making some NPCEditor quality of life improvements. In NPCEditor's Preferences, turn off "`Save files automatically if application is idle for 15 seconds.`"

Creating a New Character

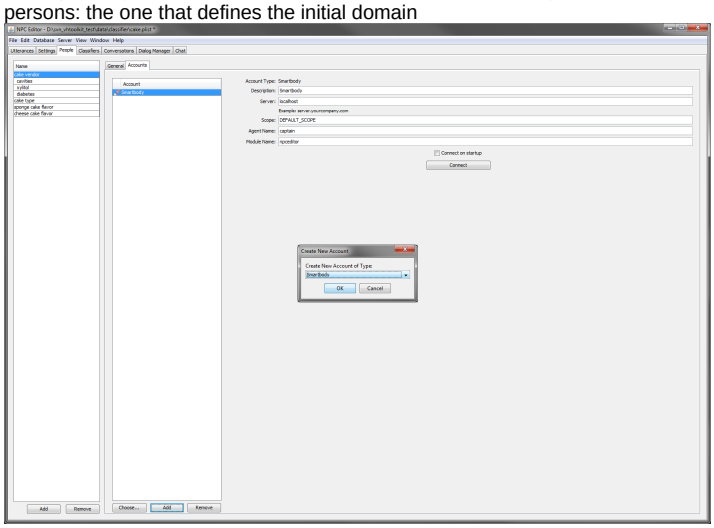
1. Select '`File->New`' to create a new plist. Save it as '`cake.plist`'.
2. Inside `cake.plist`, select the People tab and create a new person named '`cake vendor`'.This person will handle the initial greeting exchange.

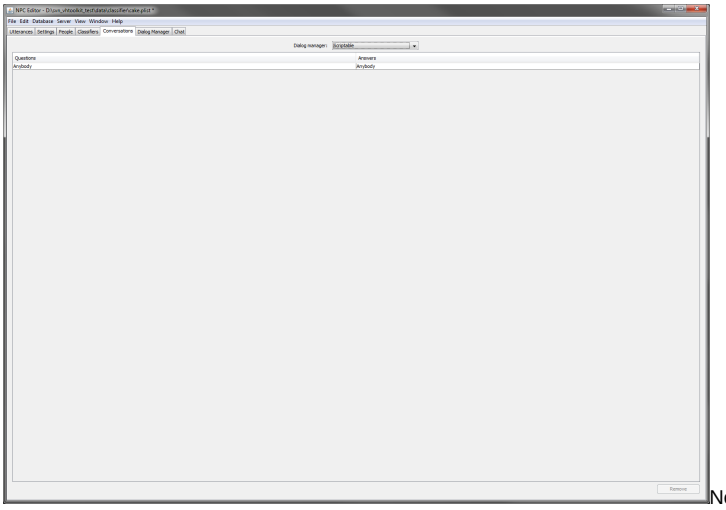


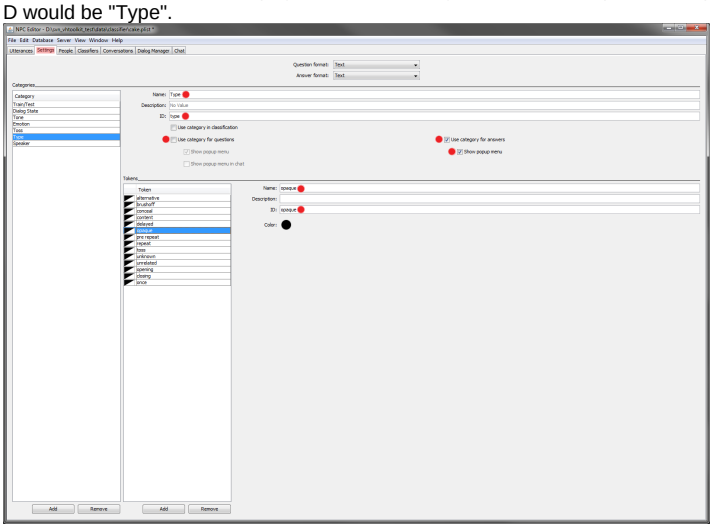
> **A note on "People" in NPCEditor**
>
> Each "person" in a plist represents a domain, which contains the list of answers the system chooses from. Domains can have inheritance, so for example greetings could always be accessible, while other details might only be available at specific points in a conversation.

Make sure you set both 'first name' and 'last name' for your person. If your person/domain's name is just one word, you can set the last name or the first name to be a space or you can split the word into 2 parts, one for the first name and the other for the last name, such as '`CakeVendor`' or '`Cake Vendor`'

3. Now define the connection NPCEditor uses to communicate with the rest of the Virtual Human Toolkit architecture. Select the 'Accounts' tab, then click on 'Add' and select 'Smarbody'. Test the connection by clicking on 'Connect': the button will change to 'Disconnect' and the row corresponding to the NPCEditor in the Launcher will become green. This should be defined once and associated only to one of the defined persons: the one that defines the initial domain



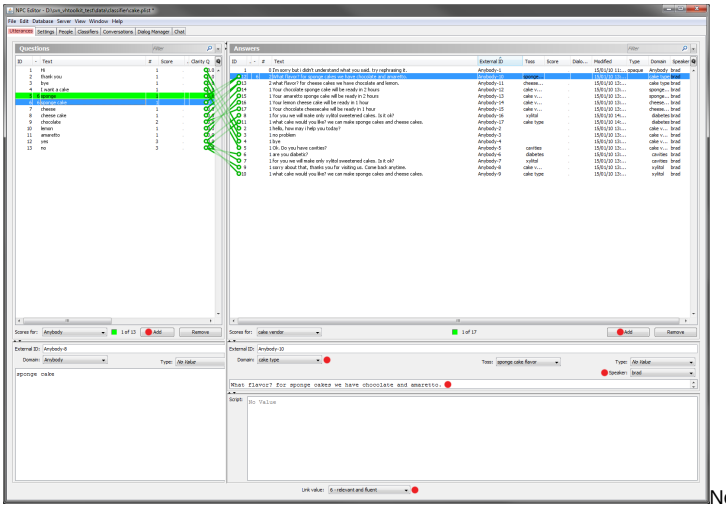
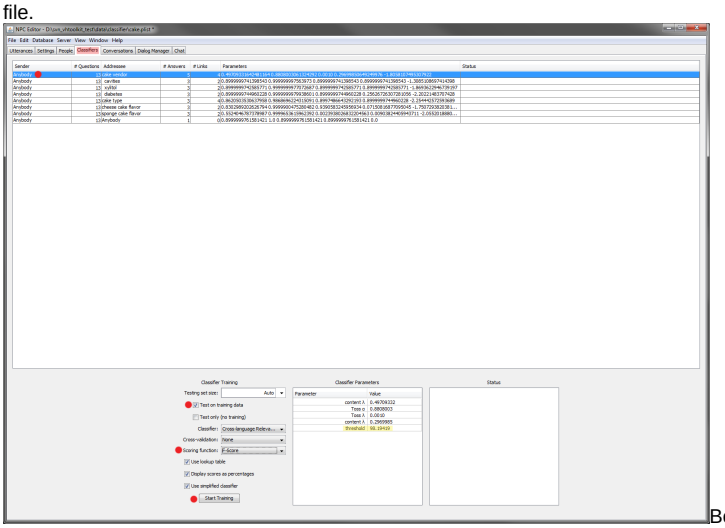
4.  Next select '`scriptable`' as the type of dialog manager in the 'Conversations' tab. A new 'Dialog Manager' tab next to 'Conversations' will appear containing an initial script for a dialog manager. The script is written in Groovy and can be edited to suit your needs.
5. Set the parent property of '`cake vendor`' to '`Anybody`' (default value). This step defines an inheritance hierarchy among the various domains. In this case the '`cake vendor`' domain inherits the utterances defined for the '`Anybody`' domain.
6. Create all the other persons defined in the dialog graph above: '`cavities`', '`xylitol`', '`diabetes`', '`cake type`', '`sponge cake flavor`', '`cheese cake flavor`'. Set the parent property of each of them to the '`Anybody`' domain.
7. Create a '`Type`' and '`Speaker`' category in the 'Setting' tab. The first is used by the default dialog manager script to handle off topic utterances from the user. The second (speaker) needs to be set for proper communication with the rest of the modules in Virtual Human Toolkit. Note that the DM uses the ID of a category for selection, so you need to change the autogenerated key to something unique, such as the Name, so Type's ID would be "Type".



8. Make sure the '`Toss`' category is set to be included in the answers and used by the classifier. This completes the setup, and we can move on to creating new content.

Adding Content to your Character

1. Now on we will add the utterances we want the agent to understand and we will define the appropriate answers. To define the reply the agent will give when it doesn't understand what the user said; add a new answer and select the type to be '`opaque`'. Then define an appropriate text (e.g. "`I didn't understand what you said, try to rephrase it`"). For all answers you want the agent to be able to speak and animate, set the speaker to '`Brad`'.

2. 
Next add your dialog. Each utterance should be the agent's answer to a user's statement, such as 'Hi'. For the 'Hi' example, leave the type of the answer unset (because it's not an answer to be associated with non understanding, but an opening greeting). Set the speaker to 'Brad' and the domain to 'cake vendor'. On the user pane (the left half), add a new utterance with the text 'Hi'. Finally click on both newly added utterances (they both become blue) and set the link strength to '6' to specify that the two greetings are a question/answer pair. Similarly do for adding the replies for 'thank you' and 'bye'.
3. As an example of tossing from one domain to another, setup a toss from 'cake vendor' to 'cavities' when the user says they want a cake. To do so, Add a new utterance as a resppnse, and set the toss property to the 'cavities' domain.
4. Follow this procedure and complete all steps as represented in the dialog graph given at the beginning. You can see also the complete cake.plist file.
5. 
Before running the example, you will need to train the classifier. This analyzes the links between user and agent utterances, and defines how the agent will respond. Select all the rows in the table in the tab 'Classifiers'. Then check the box 'Test on training data' (because the training data is too small to be split) and then click on 'Start Training'. You can change the optimization criteria. Also, in this example, the 'Anybody' classifier will fail because it contains no links to learn between user utterances and system answers.
6. :If during the testing of the example you see that the classifier returns multiple answers instead on just the correct one, you can adjust the threshold of that classifier by selecting the classifier in question and double-clicking the threshold displayed.

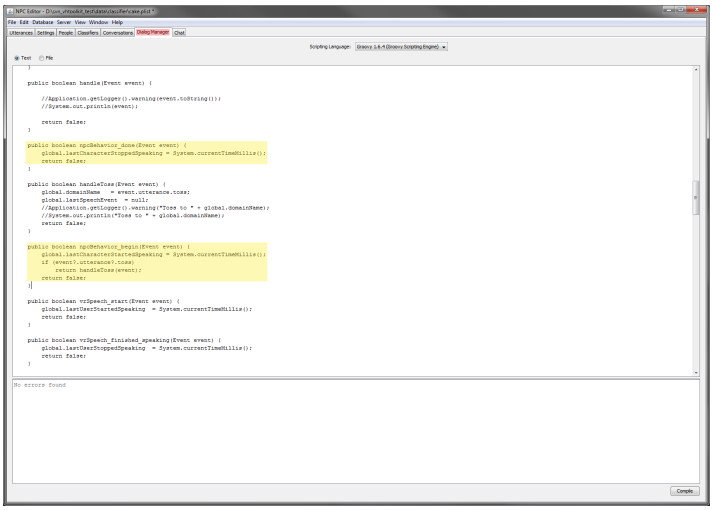
Making The New Example The Default One

After creating a new example and saving the plist, to make it load automatically every time the NPCEditor runs, you'll need to edit the config for theVirtual Human Toolkit launcher. Editing the NPCEditor launch script that in `svn_vhtoolkit/trunk/tools/launch-scripts/run-toolkit-npceditor.bat` and then re-compile `vhtoolkit.sln, or, if` you want to make this change without having to re-compile, edit the file `bin/launcher-scripts/run-toolkit-npceditor.bat`.
In that script change the pointer from the default plist file to your new plist file. Also set the option 'Connect at startup' in the 'People' tab for the person /domain associated with the Smartbody connection.

Note On When The Toss To A New Domain Occurs

The tossing to a new domain is decided in the dialog manager script (that can be seen in the dialog manager tab).

The default script tosses after the character has finished speaking the complete utterance. This can cause problems when the user interrupts the agent, as their question will still be in the first domain. To modify the default behavior and move the toss from the end of the utterance to the beginning, move the two selected lines from the method `npcBehavior_done` to the method `npcBehavior_begin` as displayed in the following image:



Note On Debugging

To debug the dialog manager script one can add logging instructions. To do that one has to add this import: `import com.leuski.af.Application;` a nd use the following expression to log something: `Application.getLogger().warning(a.toString());` where a is the object we want to print in the log.

The log is saved in `$HOME/AppData/Local/Temp/edu.usc.ict.npc.editor.EditorMain.0`

If you are unsure on the location of the log file, you can use tools like Process Explorer.

An additional way to debug it is by using the debugging capabilities of an IDE. With IntelliJ is easy to import the project directly from the source checked out of the svn repository. It figures out most of the dependencies (but not all). Just try to recompile all projects, and resolve one by one the errors the compile gets (that are unresolved symbol errors) by including in the dependencies of the project, the module that implements the symbols (classes) unresolved.

The `main` method to run to get the NPCEditor is the one in the class `trunk\core\NPCEditor\editor\src\java\edu\usc\ict\npc\editor\EditorMain.java`

IntelliJ can also debug Groovy together with Java (the dialog manager script is written in Groovy).

To recompile NPCEditor just run `ant` from the directory `trunk\core\NPCEditor`

If you use this way to debug NPCEditor, you may want to disable the NPCEditor row in the launcher so that only the instance started from IntelliJ is present.

Note On A Different Approach To Keep The Dialog Manager State

Another way to handle state changes is by keeping track of the state in the dialog manager script.
When the user says or types something, the classifier receives it and returns the list of most appropriate answers.
This list is what the expression `List <Map<String,Object>> answers = engine.search(global.domainName, event);`
returns (near the top of the method `public boolean vrSpeech_asr_complete(Event event)`. Each answer is an object of form `Map<String, Object` .

Within the script itself, one can keep a state variable, then the state can be changed based on the list returned by the classifier (i.e. `answers`) and a particular reply can be sent to the virtual agent.
To send a particular reply, we can change the value associated with the key 'ID' of the element in `answers` that we want to send to the virtual agent for speech production and animation.
Each answer in the 'Utterances' tab in the NCPEditor has an 'External ID' column. So, to send the utterance we want, just get it's ID (that is the value of the 'External ID' column, this should have be manually stored in the state variable), then pick one of the objects in `answers` , change the value associated with the key 'ID' to the 'External ID' selected and send the modified object using the `send` method.