Deus Ex

Software Development Kit
Conversation System

09/07/00

By: Sheldon Pacotti, Al Yarusso, Scott Martin

Building Conversations with ConEdit

I. Overview

Before getting into the nitty-gritty of exactly how to add conversations and barks to the game, you might want to read over the following bird's-eye view of ConEdit, the Deus Ex "conversation editing" program. It will give you a sense of how the program is structured and what it can do.

Conversations in Deus Ex are stored in several files ending with a .con extension. There are often several files in use per mission in the game, but most commonly there are two: one file for the main conversations and then a file for the InfoLinks. The naming convention is generally "Mission1.con" and then "Mission1_Infolink.con". Some conversation files are shared by many missions, such as "AIBarks.con", which is the file that contains all the AI Barks. (These are "barks" triggered by the AI routines, such as Idle, Searching, Attacking, Critical Damage, etc.)

When you open a conversation file, the left pane shows a list of all the NPCs who have conversations in that file. For instance, in Mission1.con in the left pane there'll be a "PaulDenton" item. If you expand this item, you'll see a list of Paul Denton's conversations. One of those conversations is "MeetPaul". If you click on that conversation, you'll see a list of all the conversation's "events" in the right pane. Some conversations also have flags associated with them. In those cases, you can expand the conversation (in this case, "MeetPaul") to see what flags are required to be set before the conversation will trigger.

In the right "Conversation" pane, all the events for the conversation are listed. Most of these will be Speech events, which are the lines of dialogue that are spoken by NPCs and JC Denton. You can double-click on any of these events, and a dialog box will appear with more detailed information regarding the event. In the case of the Speech event, you'll see the speaking actor, the person being spoken to, and the actual speech. There is also an edit box that contains the name of the audio file that will be used for this line.

There are several different types of events, and each is displayed differently in the conversation pane. In addition to the Speech event, some of the other common events are:

- Choice: Allows JC to make a choice
- Move Camera: Controls camera placement in the scene
- Comment: A comment for the voice actors (usually)

- Set Flag: Allows one or more flags to be set
- Check Flag: Allows one or more flags to be checked, and then jumps accordingly
- End: Ends the conversation
- Jump: Jumps to another Label in this or another conversation

Labels are used to control the flow in a conversation. Any event can be given a label: simply double-click on the event and type in the label in the Event dialog box. Labels are displayed in the left column of the conversation pane.

Several events allow you to jump to a label in the conversation. The most obvious is the "Jump" event. But there are several others that also allow jumping to labels, such as the "Set Flags" and "Check Persona" events.

II. Compiling Conversation Files

Now for the specific details.

The only tricky part about adding conversations to a map is creating the .con files and updating various other text files so that your new stuff gets compiled correctly. The first step is to copy all of your .con files into the c:\DeusEx\DeusExConversations directory (assuming the game is installed in C:\DeusEx). Next, you need to tell the compiler to import them into the game. You do this by creating a text file called DeusExConversations.uc in the c:\DeusEx\DeusExConversations\Classes directory. (Create the directory and the file if they don't exist.) You must add lines to this file that look like this:

```
#exec CONVERSATION IMPORT FILE="Mission0_Infolink.con"
```

where the text in quotes is the name of a given file. Add a separate #exec line for each .con file.

Once you have the .con files copied over and have told the compiler to import them, you're ready to compile. Before running ucc, first delete all of the conversation .u files in the System directory. You can do this by typing the following at a DOS prompt with C:\DeusEx\System as the current directory: "del DeusExCon*.U". Now type "ucc make" to run ucc. That should do it. Remember to quit UnrealEd before compiling, or the compile will fail. If you followed the instructions in Section VI: Recording Conversation Audio (below), the audio should get built into the game automatically. Expect audio compilation to take a few minutes.

To save time, you probably want to create a batch file to do some of these steps for you. The DOS batch file below copies .con files from a working directory into the DeusExConversations directory, deletes the old .u files, then rebuilds the game. Run it from C:\DeusEx.

```
@echo off
echo .
```

```
echo Copying conversations
copy "c:\Conversations\Mission0.con" c:\DeusEx\DeusExConversations
copy "c:\Conversations\Mission0_Infolink.con" c:\DeusEx\DeusExConversations
echo .
echo Compiling new conversations...
echo .
del System\DeusExCon*.U
System\ucc make
```

III. FNames

Before we go further, a word should be said about naming. Many of the "names" you will define for NPCs or conversations will be stored, internally, as FNames. Flag names, conversation names, NPC bindnames, object names, Goal names, and so on are all stored as FNames. If in doubt, always make sure the names you enter into ConEdit and UnrealEd conform to the limitations of an FName. An FName is a C++ class built into Unreal that has the following restrictions:

(a) The name can contain no spaces.
(b) The name can contain "_" characters but no other punctuation.
(c) The name should begin with a letter, not a number.

This is why the conversations for an NPC called "Anna Navarre" are associated with the internal bindname "AnnaNavarre".

Deviating from these restrictions may not crash the game, and ConEdit may not complain (in some cases) if you break the rules, but bad names could lead to unexpected behavior. Be safe and commit the FName restrictions to memory. From here out, the data type will be referred to by name without reviewing the specifics.

IV. Creating Conversation Files

To create a specific conversation for a specific NPC, you need to carefully construct a .con file to go with a particular map. When you create a new .con file (File → New), a Properties dialog box appears which lets you specify the mission(s) in which this .con file will be active. (Yes, you can only select the mission numbers in the list, which correspond to the Deus Ex missions.) A .con file will be "active" on all maps whose "mission number" matches the number(s) of the mission(s) you select here. (A map's mission number is specified in the DeusExLevelInfo object: see the Editor Documentation). In other words, each map is defined to be part of a single mission (1, 2, 3...), but each .con file can be made available to multiple maps and/or multiple missions.

Now, let's assume you've created a .con file and associated it with a particular mission. You can now create conversations of three types, which will be covered below. Each conversation has an "Owner," which roughly corresponds to the NPC in the world who will be the speaker and/or trigger of the conversation. You define the owner in the "Conversation Owner" field of the Conversation Properties dialog box, which appears when you create a new conversation (Conversation → Add Conversation). The value you enter in this field must be an FName (described in Section III, above) and must

match up to the "conversation bindname" of an NPC in the world. (The process of dropping and naming an NPC will be discussed below.) Each conversation also has a "conversation name," which like the NPC bindname is an FName.

Here are the three kinds of conversations you can create:

Infolinks

(1) To add infolinks, first create a file called "MissionX_Infolink.con." (This is just a convention for the sake of organization, and the "_Infolink" part of the name is not mandatory. In theory, you can mix infolinks and normal conversations in a single file, but this was never done in the original Deus Ex.)
(2) Associate this file with a particular mission with the Properties dialog box, as described above.
(3) Create a new conversation (Conversation → Add Conversation), and set the Conversation Owner to "JCDenton" (the player).
(4) Under the Options Tab of the Conversation Properties dialog box, click the "Datalink Conversation" radio button. Most of the other radio buttons will be grayed out, and this means you have a conversation that can be triggered as an infolink.
(5) The convention in Deus Ex is to precede the name of the conversation with "DL_" -- i.e. "DL_TalkToMe" -- but this is not mandatory. (Infolinks used to be called datalinks.)
(6) In an infolink conversation, camera-movement will be ignored, but you can check logic, add multiple Speech events, add/clear goals, and do most everything else. (see Section V: "Conversation Events").
(7) To trigger the infolink on a map, drop a DatalinkTrigger (see the Editor Documentation).
(8) NOTE: Deus Ex only has thumbnail "talking head" images for certain primary NPC's, so most of your infolinks will probably come up with a generic icon.

AI Barks

You probably want to put all of your AI Barks in a file called AIBarks.con, but this is not mandatory. You can also create multiple .con files that are active during different sets of missions. Such an approach was necessary in *Deus Ex* to deal with NPCs like Walton Simons who are allies for several missions and then become your enemy.

You create a pool of AI barks for a particular game-state by putting them in a single conversation with a name that corresponds to that game-state and to the NPC that will be doing the barking. For instance, the following is the list of Anna Navarre's AI-bark conversations. The Conversation Owner of each of these conversations must be "AnnaNavarre". (NPC's "own" their AI barks.) Also, the NPC's bindname must appear in the names of the conversations exactly as the substring "AnnaNavarre" does below. Since these conversations are barks (i.e. not blocking cinematics), you need to check both of the following fields under the Options tab of the Conversation Properties dialog: "Non-

interactive Conversation" and "Remain in First-Person Mode." These fields are discussed in more detail in the "NPC Conversations" subsection.

AnnaNavarre_BarkAllianceFriendly (player attacked an ally, and the ally decided to stop fighting and be friends)
AnnaNavarre_BarkAllianceHostile (player attacked an ally, and the ally decided to fight back)
AnnaNavarre_BarkAreaSecure
AnnaNavarre_BarkCriticalDamage
AnnaNavarre_BarkFutz (player messes with something in the world which has an Advanced → bOwned property = true)
AnnaNavarre_BarkGoingForAlarm
AnnaNavarre_BarkGore (NPC discovers corpse)
AnnaNavarre_BarkIdle (generic idle noise: cough, whistle, muttering)
AnnaNavarre_BarkOnFire
AnnaNavarre_BarkOutOfAmmo
AnnaNavarre_BarkPostAttackSearching (enemy NPC was fighting the player but lost contact)
AnnaNavarre_BarkPreAttackSearching (enemy NPC heard a noise)
AnnaNavarre_BarkPreAttackSighting (enemy NPC sees the player)
AnnaNavarre_BarkSearchGiveUp
AnnaNavarre_BarkSurprise
AnnaNavarre_BarkTargetAcquired (enemy NPC ready to attack)
AnnaNavarre_BarkTargetLost
AnnaNavarre_BarkTearGas

Inside each of these conversations, you would use the Random event to randomly select from a list of Speech events (see Section V: "Conversation Events"). In pseudo-code, the conversations would look like this:

```
RANDOM
        B1
        B2
        B3

B1  SPEECH: "Agh!"
END

B2  SPEECH: "Ow!"
END

B3  SPEECH: "Ooof!"
END
```

NPC Conversations

Though this information is covered elsewhere in the docs, I'll be expansive and walk you through the process of dropping an NPC in the world, giving the NPC a name, and finally giving that NPC something to say.

Add an NPC to the World

(1) You place people in the world just like you do other objects. Highlight the name of a person (called a pawn or an NPC) in the class browser in UnrealEd. (All of them are contained in Pawn::ScriptedPawn.)

(2) Next R-click on the floor (in the 3-D view) where you want the person to start and select "Add X here," where X is the name of the person you selected.

(3) You tell the person what to do with "Orders". Select the Properties of the person and go to Orders → Orders. You actually have to type in a keyword here -- there is no menu. Type "Wandering" to tell the person to wander around aimlessly. Type "Standing" to tell the person to stand in one place. (See the Editor Documentation for a complete discussion of orders.)

(4) Name the person by setting the fields inside the Conversation properties. The property Conversation → BindName is the most important part of the name. It's an FName (no spaces, no punctuation except "_") which ConEdit uses to identify a given person. A character named Paul Denton has a BindName of PaulDenton, for instance.

(5) Use the properties Conversation → FamiliarName and Conversation → UnfamiliarName to define the real name of a character. A janitor named Bob might have an "UnfamiliarName" of "Janitor" and a "FamiliarName" of "Bob". The names you type here are used only as text identifiers for speaking characters. They are unimportant compared to the BindName, and they can have spaces in them.

(6) Now you have a person in the world who can wander around and talk to the player.

Give an NPC a Conversation

Open the .con file associated with a particular mission in ConEdit. Select the menu option Conversation → Add Conversation. The Conversation Properties dialog will appear. In the field labeled "Conversation Owner," type the BindName of the NPC who needs something to say (discussed in the previous section).

Next you need to choose what kind of conversation this will be (bark, cinematic, uninterruptible; etc). Go to the Options tab. Despite the number of radio buttons, you really only have four meaningful options:

(1) "Display Conversation Only Once" -- Uncheck this if you want the convo to play multiple times (i.e. for a group of barks you want to have play over and over).

(2) "DataLink Conversation" -- covered above. Use this to specify that this conversation should play as an "Infolink."

(3) "Non-interactive Conversation" and "Remain in First-Person Mode" -- check BOTH of these boxes to create a "non-blocking" conversation, one that plays in PoV mode without stopping the game (i.e. barks and overheard conversations). Checking only one or the other of these fields doesn't make much sense and isn't very useful. By checking NEITHER of these fields, you can create a cinematic conversation that will stop the game and let you set camera-angles.
(4) "Absolutely CANNOT be interrupted by another conversation" -- This does what it says, but due to last-minute engine-tweaks the "Can Be Interrupted by Another Conversation" doesn't do anything. By default, all Deus Ex conversations can be interrupted.
(5) "Random Camera Placement" -- We didn't use this in the game, and in fact it was never fully implemented.

Conversations can be triggered in many ways. You can even launch them via mission-scripting or with the "ConversationTrigger" (see the Editor Documentation). For now, let's look at the options available in ConEdit under the "Invoke" tab of the Conversation Properties dialog:

(1) "PC Frobs NPC" -- checking this will cause the convo to trigger when the player clicks on the NPC who owns it.
(2) "NPC Enters PC Radius" -- checking this will cause the convo to trigger whenever the player gets within a certain radius of the NPC. You define the radius in the text field beneath this radio button. The integer value you enter should be in Unreal "units". 16 units = 1 foot.
(3) The other two options don't do anything. The "seeing" option was never implemented and the "Bump" triggering was goofy and got hard-coded to behave like frob-triggering.

Optionally, you may want a conversation to trigger only during a particular game-state. To do this, go to the Flags tab (in the Conversation Properties dialog) to add a list of flag-names. All flags in Deus Ex are Boolean, so you can only check for True and False. By double-clicking on a flag in the list, you can toggle the check between True and False.

V. Conversation Events

The actual dialogue and logic of a conversation is defined with a series of "events," which appear in the right-hand pane of ConEdit. You can add and insert events by R-clicking in that pane to bring up a menu. Typically, you want to "add" an End event (to mark the end of the conversation) and then "insert" the rest of the events at particular locations.

Most of the events that you create will be Speech events, which contain the dialogue spoken by NPCs and JC Denton. You will also need Camera events (to move the camera in cinematic mode), Choice events (for branching), events for checking/setting flags, jump events; etc. The various kinds of events are described in the alphabetical listing below. To access an event's properties in ConEdit, just double-click on it. The one

property that all events share is the "Event Label," which is an alphanumeric value (an FName) analogous to labels in Pascal or C++. You can "jump" directly to any label, either as the result of a logic-check or a player choice.

A conversation's list of events is processed in order until the conversation engine reaches an End event or the end of the list. In this way, a conversation is like a simple computer program. You can think of the events as commands.

IMPORTANT NOTE: When you add or edit an event, you have to press "Update" in order for your changes to get added to the conversation. If you press "Close", the Properties dialog will simply disappear and your changes will be lost.

Here is an alphabetical listing of the various events, including descriptions of the Properties dialog associated with each event.

Add/Complete Goal -- Use this event to create or clear a goal. The "Goal Name" is a unique FName value used internally to identify the goal. The "Goal Text" box contains the text description that the player will see. Checking the "Primary Goal" box will make the goal "primary," which means it will remain with the player until it is explicitly cleared in another conversation. ("Secondary" goals automatically vanish from the Goals/Notes screen at the end of each mission.) Click the "Goal Completed" box to specify that the goal has just been completed. When completing a goal, you only need to specify the Goal Name, not the description.

Add Credits -- Adds credits to the player's money supply. To subtract credits, specify a negative number.

Add Note -- Adds a Note to the Goals/Notes screen. Useful for recording critical information such as door-codes, so that the player doesn't have to write anything down.

Add Skill Points -- Gives the player some skill points.

Buy/Sell/Trade -- not implemented.

Check Flag -- Jumps to a particular label based on the values of a group of flags. You specify the label in the "If Flags Set Jump To" field. The label must exist in the current conversation. To define the logic that needs to be checked, first use the "Add Flag" button to bring up a list of available flags. If the flag you want to check is not in the list, click the "Edit Flags" button to add a new flag-name to the table, then return to the Flag List dialog and double-click on the flag you just added. Back in the main Properties window, you can double-click on a flag to toggle whether its value should be true or false. Note that the Check Flag event will perform a logical AND operation on the flag values you specify; that is, the Check Flag event will perform the jump only when ALL of the flags you listed have the values you specified. To simulate an OR operation, use two Check Flag events in a row.

Check Object -- This event will jump to a label when a particular object is NOT in the player's inventory. The "Object To Look For" field will let you select an object from the .con file's table of objects. To add an object to the table, just type its class name into the field and then click to edit another field in the Properties dialog. (You will automatically be asked whether you want to add the new object to the table.) Examples of class names: WeaponModSilencer, WeaponPistol, Binoculars. (These are the class names that appear in UnrealEd's class browser.) Specify the destination label in the "If Object Not Found, Jump to Label" field. The label must be in the current conversation.

Check Persona -- Use this event to perform a logical check on the player's health, credits, or skill points. You can specify a logical operation in the "Condition" field, a value in the "Value" field, and the destination label in the "Label to jump to" field. The label must be in the current conversation.

Choice -- This event presents the player with a dialogue choice. The "Add Choice" button will bring up the complicated "Edit Choice" dialog, which will allow you to specify a single line of dialogue and some logic associated with it. You have to specify at least two dialogue options. When the player clicks on one of the options, JC will say the line, then the conversation will jump to the label associated with the particular choice. If you put a Move Camera event just prior to a Choice, the engine will use that camera angle when JC speaks the choice text. The choice text and destination label are defined together within the "General" tab of the "Edit Choice" dialog. The "Flags" tab lets you specify that the dialogue option appear only when certain logical conditions are met. For instance, if you wanted JC to be able to ask about the NanoSword only after the player has heard an NPC talk about it, you could use the Flags tab to check for KnowsOfNanoSword = true. Then, in another conversation somewhere, you could use the Set Flag event to set KnowsOfNanoSword = true when an NPC mentions the sword.

Comment -- Use this event to make notes to yourself or for the voice-actors. The player won't see this text.

End -- Ends a conversation. It's like an exit() command in a C program. A conversation can have multiple end-points.

Jump -- Jumps to a particular label. In addition to the "Jump to Label" field, there is a "Conversation" field, which means that this event can perform a jump into another conversation.

Move Camera -- This event lets you specify a particular camera-angle. The only field guaranteed to do anything is the "Predefined Camera Angle" field, which contains a list of standard camera-angles like close-ups and shoulder-shots. The engine will try to use the angle you specify, but if the camera collides with a wall the camera will be moved to a "fallback" position. Camera transitions, positioning, and random placement have not been implemented.

Play Animation -- not implemented.

Random -- A very useful event for barks. The Random event contains a list of labels from the current conversation. By default, it randomly jumps to one of these labels every time it is executed. Alternately, you can click the "Cycle Events" box, which will make the Random event jump to the labels in sequence: it will "cycle" through them indefinitely. Cycling through barks can be useful for giving an NPC a train of thought without stopping the game with a cinematic. There are two additional options for cycling. (1) "Only Cycle Once" -- cycles through the labels once, repeating the last one indefinitely. (2) "Random after Cycle" -- cycles through the labels once, then jumps to them randomly thereafter. ConEdit lets you check the boxes for both of these options at the same time, but obviously the behavior will be undefined if you do that. A good way to use Random events is to have one at the beginning of a bark-conversation and one at the end. The one at the beginning can "cycle only once" through a specific train of thought, ending with the label attached to the second Random event. The second Random event can be completely random and contain only labels for a few generic barks. In this way, an NPC can dump a bunch of information on the player, then end up in a randomized but steady state at the end.

Set Flag -- Sets the values of one or more flags. Use the "Add Flag" button to bring up a list of available flags. If the flag you want to set is not in the list, click the "Edit Flags" button to add a new flag-name to the table, then return to the Flag List dialog and double-click on the flag you just added. Back in the main Properties window, you can double-click on a flag to toggle whether its value should be true or false.

Speech -- All dialogue is contained inside Speech events. The "Actor to Speak," "Speaking To," and "Speech" fields must all be defined for each line of dialogue. The first two contain BindNames of NPCs (or "JCDenton"), while the Speech field contains the text of the line to be spoken. An audio file can also be specified, but since audio filenames are generated automatically, you probably shouldn't mess with this field.

Transfer Object -- Transfers an object from the player to an NPC or vice-versa. You can also transfer items between NPCs, if you want to get creative. You can select the object to transfer from a pull-down menu, or you can type the name of an object that isn't in the table. By clicking on another field, you'll automatically be asked whether or not to add a new object to the table. Usually, you only want to transfer one object, but in the case of grenades, multi-tools, lockpicks, and other "stackable" items, you can specify that more than one be transferred at a time. This feature was added to fix various bugs, so if you want to transfer 6 grenades to the player, do it with one event. However, don't try to transfer multiple pistols at the same time -- they aren't stackable. Now... in case you thought buy/sell/trade was going to be fun, notice the "On Fail Jump To" field at the bottom of the Properties dialog. You'll need to use this field to specify a label where the conversation can jump when the player's inventory is full. Transfers won't fail for any other reason, because if you try to transfer an object that doesn't exist the engine will automatically instantiate it. Whenever you want an NPC to give the player something, you'll have to create an intermediate conversation-branch for when the transfer fails and the player has to drop stuff and then re-initiate the conversation.

Trigger -- This event simply launches a trigger that has an Event → Tag property that matches the value you put in the field "Trigger Tag." The Trigger event is reliable only when it comes at the very end of a conversation. It's useful for launching Orders triggers, unlocking doors, changing the alliance of NPCs; etc. For more on adding triggers to a map, see the Editor Documentation.

VI. Recording Conversation Audio

To add audio for the Speech events, all you have to do is record .mp3 files and put them in the right place. (.wav files will not work with this version of the engine.) The .mp3 files must conform to the following criteria:

(a) They must be mono (not stereo).
(b) They should be 48kbps, 44kHz files, because the lip-syncing code was tested and optimized for those conditions.

Compilation will happen automatically when you run ucc. (For more information on compiling, see Section II: Compiling Conversation Files.) However, for the audio to get compiled correctly, every single audio file must conform to a rigid naming convention and directory-structure. Example:

C:
\DeusEx\DeusExConversations\Audio\Mission03\PaulDenton\MeetPaul\PaulDenton01.mp3

It's not as bad as it looks. "C:\DeusEx\DeusExConversations\Audio\Mission03" simply means that you have to put the audio in the "Audio" directory associated with Mission 3. "Mission03" is actually the name of an audio "package." In ConEdit, you define which audio package a .con file uses with the "Misc" tab inside the Properties dialog. Before recording audio, be sure that all of the .con files are set up to use a particular audio package. Breaking the audio into separate packages will allow you to test one mission's conversations without having to rebuild the whole game. However, using multiple audio packages is not required.

The rest of the filename corresponds to how the conversation is structured in ConEdit.

Remember that ConEdit groups conversations together underneath the BindName of a character. Each character "owns" his/her conversations. Likewise, in a given audio package, each character has a folder that contains all of the audio for his conversations, (i.e. the "C:\DeusEx\DeusExConversations\Audio\Mission03\PaulDenton" folder above.)

Inside of the PaulDenton folder, there is a folder for each conversation that Paul Denton owns, and inside THOSE folders are the actual audio files.

In other words, the directory "C:\DeusEx\DeusExConversations\Audio\Mission03\PaulDenton\MeetPaul" contains all of the audio files for the "MeetPaul" conversation.

The files themselves are named in two parts: the BindName of the character speaking and a number that starts at "01" and counts upward, indicating how many times a given character has spoken. For the following conversation, called "MeetPaul",

JC Denton: Hi.
Paul Denton: Hi.
JC Denton: How are you?
Paul Denton: Confused.

the corresponding filenames would be:

PaulDenton01.mp3 ("Hi.")
JCDenton01.mp3 ("Hi.")
PaulDenton02.mp3 ("How are you?")
JCDenton02.mp3 ("Confused.")

Notice that the numbers are character-specific. They say basically "This is the second line of dialogue for JC" or "This is the first line of dialogue for Paul."

VII. Tips/Tricks

You now know enough to start adding conversations to a Deus Ex mod. However, there are a few miscellaneous quirks you might want to be aware of.

Bugs -- ConEdit is not entirely bug-free. Save your work often. In particular, pressing Return in a few rare instances can cause the program to crash. Typically, this happens when you've been mucking with one of the internal tables and editing an event at the same time. Just be careful!

Colors -- Events appear in different colors to make it easier to scan over a conversation. The most important color is red, which indicates that an event is trying to jump to a label that doesn't exist. This feature is fully implemented except for the Check Persona event, which won't highlight red when it contains a bad label. Just something to keep in mind.

Cut/Paste -- It's often convenient to cut-and-paste events, but be aware that if you paste an event that has a label, ConEdit may become confused about whether or not that label even exists. You can fix the problem by opening the event that uses that label and pressing the "Update" button.

Good luck!