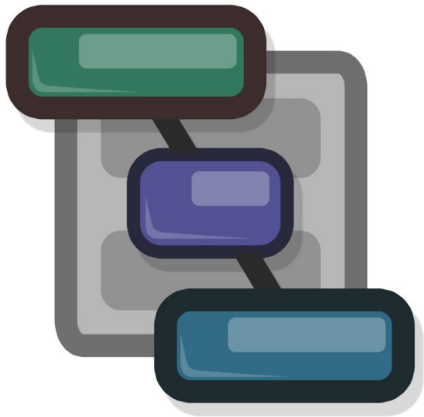
Versatile Interactive Dialogue Editor (VIDE)



By Christian Henderson
Blog: https://videdialogues.wordpress.com/
Visit the blog for news, online documentation, bug tracking, support, and donations

Index:

What is VIDE?

VIDE (Versatile Interactive Dialogue Editor) is a free plugin that simplifies the creation of complex, interactive dialogues by providing the user with a simple Player-NPC node connection interface. It does not provide the user with any dedicated in-game dialogue interface, as it is, in fact, designed to be adapted to any custom dialogue interface and communicate with it. VIDE organizes data from your created dialogues and presents it to the coder in a friendlier way. Regardless, it does provide the user with a couple of examples that cover all of its available features implemented in an actual dialogue interface.

Some of the key features VIDE includes:

- VIDE Editor: Simple, yet powerful node-linking interface.
- **[NEW]** Create Action nodes in the VIDE Editor to easily call your in-game methods.
- Go complex and connect nodes however you desire, then modify the flow of the conversation while in-game.
- Be creative! The system is not limited to conversations only.
- Start where you want: Player node, NPC node, or Action node.
- Optional and limitless multiple-choice comments for Player nodes.
- Tag and add extra data to your Player and NPC nodes.
- Split an NPC's comment into chunks in a single node by using the
 tag.
- Node tree is not linear only. You can connect multiple nodes to a single node.
- Create different start points for a single dialogue, then access and modify them while in-game.
- Go to whichever node you want while in-game by using a simple function.
- Quick and simple access to all user variables and methods through VIDE_Data component.
- BeginDialogue() and Next() is all you need to gather data from the conversation.

VIDE is not compatible with Unity's WebPlayer platform or any Unity version below 5.

How do I get started?

For a step-by-step guide, check the **Usage** section of **The VIDE Editor** chapter next page.

To get started, it is highly recommended that you import VIDE into your project and check out the provided examples. They are located at *VIDE/Examples/*
Simply load the scenes and hit Play to see it all in action, then have a look at the dialogues and the scripts to understand how it all works.

Example 1: Covers player movement, interaction with NPCs, starting up conversation, checking for extra data to do special actions, NPC tags, modifying the conversation's start point, node-calling functions while in a conversation, and updating the in-game dialogue interface using Unity's new UI system.

Example 2: Covers the minimal setup required to get things working using the GUI class.

All example scripts are heavily commented so that you know what's going on. Make sure you also check the **Scripting API Reference** section of this document to get to know the variables and functions offered by the system.

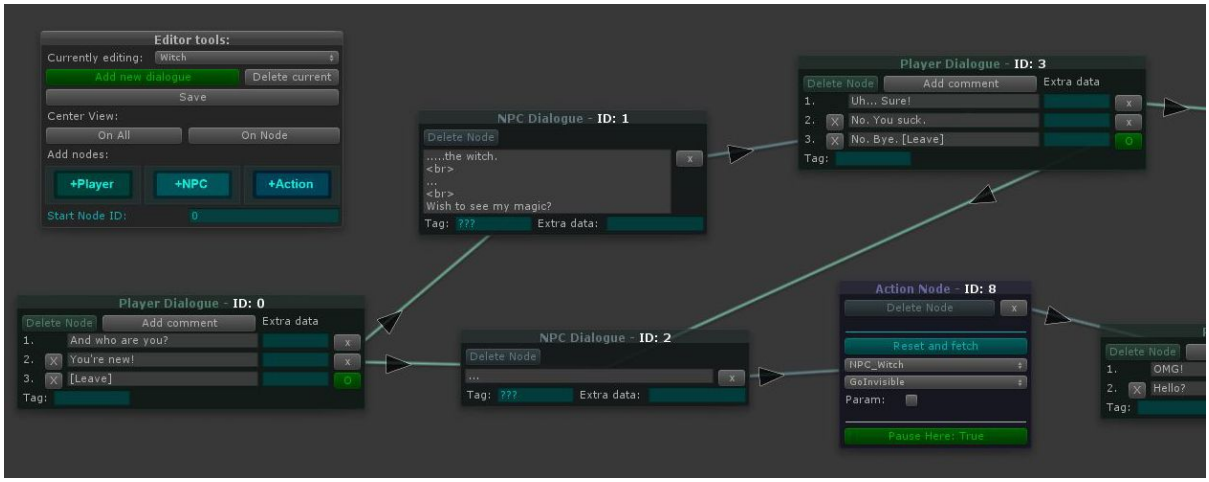
Make sure you check the **exampleUI** script. It contains various demonstrations on how to use the data that VIDE offers to modify the flow of the conversation. You can use it as a start point.

But it all starts by creating your own dialogues! You can edit the dialogues in the examples at will. Just fire up the VIDE Editor and choose the dialogue you're going to modify! See next page for reference.

IMPORTANT!
If you want, you can move the VIDE folder to another folder within your project, but to let the software know of your change, you will have to open the VIDE_Editor.cs script and edit the ***pathToVide*** *variable by adding the extra path to the new location of the VIDE folder. If you don't do this, the VIDE Editor will not work.*

The VIDE Editor

This is where you will be creating your dialogues by adding **comment nodes** and connecting them. You can connect nodes as you desire. Plus, you can create Action nodes to easily improve the dynamic of the dialogue.



The editor consists of four main window types:

1) **Editor Tools window:** Create, modify, delete, and save dialogues. From here, you also have buttons to create Player, NPC, and Action nodes, and you can specify the conversation's default start node, which can be any node.

2) **NPC comment node:** This node contains an NPC's comment. You can also fill the 'Extra data' field to pass relevant information to *VIDE_Data.nodeData,* like "item", for example. The Tag will help you identify the NPC.

3) **Player comment node:** Unlike the NPC node, you can add more than one comment to this one. Each comment will lead to a different NPC node (or to the same one).

4) **Action node:** A special node to enlist and select available methods in the scene. The method chosen will be called correspondingly. Check next section for further information.

Every node will have a unique ID that you can use to set the start point and to read from *VIDE_Data.nodeData* to meet certain conditions in your code.

Calling Next() on a disconnected node marks the end of a conversation.

Action Nodes

VIDE 1.1 introduces a third type of node called Action Node. With this node, you'll be able to call your different methods attached to objects around the scene. This will hugely improve the possibilities when looking to add actions mid-conversation.

How to use:

1. From the Editor Tools window in the VIDE Editor, create an action node by drag-and-dropping it into empty space.
2. If you check the dropdown buttons, you will see that they are empty. You need to press the "Reset and fetch" button in order to return a list of objects to select methods from. Once you press the button, it will reset all variables and do a search for valid gameobjects and their corresponding MonoBehaviours and methods **under certain rules (Check below)**. Through this way, you have control over the lists even when editing from another scene.
3. Once fetched, select a GameObject from the list.
4. Select an available method from the list below.
5. If the method has a valid parameter, you'll get an extra field to fill in with the desired data.
6. Lastly, you have the option to **pause** the flow of the conversation, just like it happens when reaching a Player or NPC node. If "Pause Here" is set to True, then you will require to call Next() again in order to continue. NodeData will not be changed.
7. All done! Do remember to connect your nodes nicely and to set the Start Node ID! Also, remember you can put them in a row!

Talk to the witch in example1 scene to see action nodes in action!
Tip: You can connect Player nodes to Player nodes by having an Action Node (It can be empty) in-between.

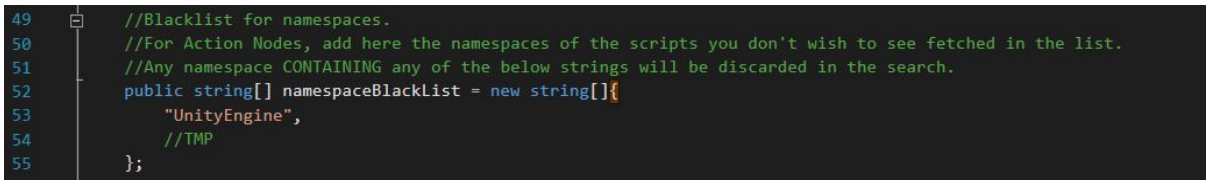
Search rules:

a. Only **enabled** GameObjects with attached MonoBehaviours will be searched.
b. MonoBehvaiours under certain namespaces will be ignored. *UnityEngine* is blacklisted by default. Check **blacklist** below for more information.
c. Declared methods have to be **public voids** and have **zero or one parameter** of type **string, bool, int, or float.**

Only GameObjects that fulfill the above requisites will show up on the list.

Action Node Blacklist:

The blacklist is thought as to filter spammy methods and objects we don't want to see in the list. Namespaces **containing** any of the keywords listed in the array will be ignored. UnityEngine is blacklisted by default as it has a bazillion "valid" methods in each of its components. You can edit the blacklist directly from the VIDE_Editor.cs script:



Close and reopen the VIDE Editor if you make a change to the list.

GameObjects with same name:

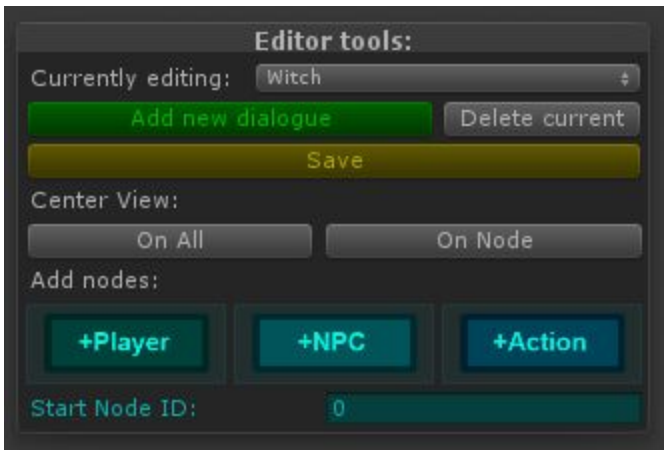
Scene objects that have the exact same name will only show up once in the object list. Nevertheless, when calling the selected method while in-game, it will attempt to call it in every object with that same name. Unless you want this to happen, it is suggested to not keep objects with same names.

VIDE Usage:

1. In Unity, go to *Window* > *VIDE Editor* to open up the editor. You can also open it from the VIDE_Assign component.
2. Click “Add new dialogue”. Name your new file and click ‘Create’. Valid characters: **a-z, A-Z, 0-9, _$#&**
3. Drag and drop a new Player or NPC node from the Editor Tools window.
4. Click and hold the Link button, then drag the cursor to an empty space and release. This will create the corresponding new node and automatically connect the current node to it. You can also release while on top of another node to connect them. If you want to connect an NPC node to another NPC node, use the “Add NPC Node” to create it, then connect the nodes. Same goes for Action nodes.
5. Continue building your dialogue. Remember that any disconnected Player or NPC comment or Action node will set the end of the conversation (isEnd = true).
6. Make sure you specify the Start Node ID in the Editor Tools window with an existing node ID.
7. Save your dialogue by clicking the ‘Save’ button.
8. In your game Scene, attach a **VIDE_Assign** component to your NPC game object.
9. In the Inspector for the VIDE_Assign component, select the desired dialogue for that NPC using the dropdown box. You can also add a custom name for the dialogue.
10. Lastly, have or add a script that will manage all UI-related stuff in your desired game object. Within the script, make sure you add a variable of type VIDE_Data, then attach a component for it. With this, you will be able to read all of the Node data to create your UI. Check the **How it works** chapter for a more detailed explanation.

Tips and good-to-knows:

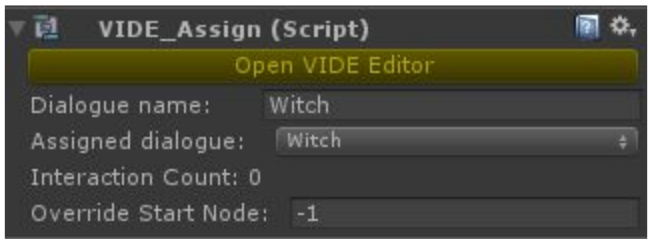
- For NPC comments, you can use the **
** keyword to split the comment into a series of comments.
- Right click on empty space to invoke the Editor Tools window.
- Click and drag on empty space to drag all of the nodes.
- Click and drag on a node to drag it, also works with Editor tools window.
- Release a connection line on empty space to create a new node and connect automatically.
- VIDE will not automatically center your dialogues when you load them to the Editor. If you can't see the nodes you've created, center the view by clicking the "On All" button in the Editor Tools window.
- You can have more than one conversation in a single dialogue file, but remember you can only have one default start point. To start in a different node instead, use the SetNode() method at the beginning of the conversation, or set **Override Start Node** field in the Inspector for the **VIDE_Assign** component to a node ID. Check the Example 1 **exampleUI** script for reference on how this can be achieved.
- You will not be able to save if you have errors in your dialogue:
    - Start Node ID is nonexistent.
- You can duplicate your dialogue and save it with a new name if you click the 'Save' button.
- **Saving**: VIDE 1.1 removes autosave feature. The VIDE Editor will save whenever it loses focus (clicking outside, closing it or Unity). It also saves when selecting a different dialogue to edit. Remember the "Save" button in the Editor Tools will turn yellow if you have unsaved changes.



How it works

Initially, you'll require 3 things to get the system working:

- Saved dialogues made with the **VIDE Editor**.
- Your custom dialogue interface with a ***VIDE_Data*** component attached.
- ***VIDE_Assign*** component attached to NPCs or to whatever game objects you want. You will need this component to start the conversation by calling BeginDialogue() on the VIDE_Data component.



The VIDE_Data component contains a variable called **nodeData**. This variable of type **NodeData** stores all of the current node's data. At the beginning of the conversation, the current node is equal to the Start Node you selected in the VIDE Editor (unless you set the Start Node Override). When you call Next() method, the conversation will go one step forward following your node structure. nodeData will then be populated with data from the next node which will be now the current one. Note that Action nodes will not modify nodeData.

Essentially, you're constantly reading the contents of *VIDE_Data.nodeData* to do whatever you want in your dialogue interface using its data. All you need is a reference to the VIDE_Data component:

```
public class dialogueUI_template : MonoBehaviour
{
    [System.NonSerialized]
    public DialogueData dialogue;
```

1. User calls *VIDE_Data.BeginDialogue()* method on the VIDE_Data component to begin the conversation with the NPC. The method requires the user to send the **VIDE_Assign** component attached to an NPC or other game object. This will populate the *VIDE_Data.nodeData* with data from the first Node that begins the conversation.
2. User uses the data in VIDE_Data.nodeData to customize the in-game dialogue interface.
3. User calls *VIDE_Data.Next()* on the VIDE_Data component to populate VIDE_Data.nodeData with the data from the next Node in the conversation.
4. User uses the new data in *VIDE_Data.nodeData* to customize the in-game dialogue interface.
5. Step 3 & 4 repeats until user reads the *nodeData.isEnd* variable to know when to call the *VIDE_Data.EndDialogue()* method and clean the dialogue interface.

It is very important that you check the **Scripting API Reference** next chapter in order to understand the methods and the contents of **NodeData** class. Once you get to know the variables and methods offered, you'll know how to incorporate them to your UI script.

You don't have to start from scratch!
Remember VIDE already comes with a fully-commented **exampleUI.cs** script that communicates with VIDE_Data. You can use it as a start point and to understand the functioning, but do remember that it has a basic design. Feel free to modify it and create any UIManager as your heart desires. VIDE will provide all data needed to make it work.

Scripting API reference

VIDE_Data

Script component attached to your dialogue UI game object. This component will store all the data regarding the currently loaded dialogue. Store a reference variable of it within your UI script to access the following:

Functions:

public NodeData **BeginDialogue(**VIDE_Assign diagToLoad**);**
Loads up the dialogue just sent. Populates the **nodeData** variable with the first Node based on the Start Node. Also returns the current **NodeData** package. If first node is an Action node, nodeData will be null until reaching a Player or NPC node.

public void **EndDialogue();**
Wipes out all data and unloads the current VIDE_Assign, raising its **interactionCount**. Do not call BeginDialogue() again if you haven't called this yet.

public NodeData **Next();**
Populates **nodeData** with the data from next Node based on the current **nodeData**. If current **nodeData** belongs to a Player Node, make sure **nodeData.selectedOption** is correctly set before calling Next(). If current nodeData belongs to an NPC Node with multiple comments (when using
), calling Next() will advance through those comments before getting to the next Node. Calling Next() on a disconnected comment will set the **isEnd** variable to true and will not get any new nodeData. Read that variable to call EndDialogue(). Also returns the current **NodeData** package.

public NodeData **SetNode(**int ID**);**
Ignores current **nodeData** state and jumps directly to the specified Node, be it Player, NPC, or Action Node. Make sure the ID exists.

Variables:

public bool **isLoaded;**
Is there a dialogue currently loaded?

public int **startPoint;** (ReadOnly)
The ID of the default Start Node set in VIDE's Editor Tools window. Returns 0 if there's no dialogue currently loaded.

public VIDE_Assign **assigned;**
Reference to the currently loaded VIDE_Assign component. Variable is null when no dialogue is currently loaded.

public NodeData **nodeData;**
Variable containing all of the current Node data you'll need to set up your dialogue interface.

NodeData class
This class stores all of relevant variables of your current node.

public int **nodeID;**
The current Node´s ID.

public bool **currentIsPlayer;**
Is this current Node a Player Node?

public bool **pausedAction;**
Are we currently on a paused Action Node?

public bool **isEnd;**
Is it the end of the conversation?

public string[] **playerComments;**
An array of strings with all of the Player node's comments.

public string[] **npcComments;**
An array of strings with all of the NPC node's comments. If not using
, the size of this array will always be 1.

public int **npcCommentIndex;**
The current index of the focused NPC comment when there're more than one. It is always 0 if not using
.

public int **selectedOption;**
The index of the currently selected player comment. When calling Next() on a Player Node, the method will read this variable to know where to go next.

public string **extraData;**
The string of extra data declared on the VIDE Editor for NPC nodes.

public string[] **playerCommentExtraData;**
The string of extra data declared on the VIDE Editor for each player comment.

public string **tag;**
The tag you set for the NPC Node on the VIDE Editor.

public string **playerTag;**
The tag you set for the Player Node on the VIDE Editor.

VIDE_Assign
Script component attached to the game objects you want to have a dialogue loaded from. Either by using **VIDE_Data.assigned** or by creating a reference to the component yourself, you can have access to the following:

Functions:

public string **GetAssigned();**
Returns a string with the name of the currently assigned dialogue (Not the custom name assigned on the Inspector).

public bool **AssignNew(**string dialogueName**);**
Assign a different dialogue to this VIDE_Assign. The dialogue you're going to assign must exist, otherwise the method will return false. Doing this is the same as selecting it from the Inspector. Do not include the file extension for the dialogue name.

public string **GetFirstTag(**bool **searchPlayer);**
Returns the first tag it finds, be it Player or NPC tag, depending on the boolean.

Variables:

public int **interactionCount;**
This variable begins on zero. Every time you call EndDialogue() on VIDE_Data while having this VIDE_Assign currently loaded, interactionCount will increment by 1. In the end, it keeps track of how many times you've interacted with this game object in particular.

public string **dialogueName;**
The custom name for this dialogue. Can be set from the Inspector.

public int **overrideStartNode;**

Default is -1. When changed, the assigned dialogue's Start Node will be ignored and will use this one instead. Make sure the ID exists. This is an in-game change only, it does not modify the actual dialogue's Start Node. Set it back to -1 to use original Start Node. You can also set it from the Inspector.

Changelog

Version 1.1

- Implemented Action nodes to call methods within scripts, including blacklist.
- Added extraData fields to player comments.
- Added *playerCommentExtraData* and *pausedAction* variables to Node Data.
- Removed "End Here" buttons. Now, *isEnd* becomes true when calling Next() on a disconnected node of any type.
- New method for VIDE_Data. *GetFirstTag()*
- Now you can Drag&Drop new nodes into the canvas.
- Added smart arrows to the connection lines.
- Removed autosave and polished save system.
- Now VIDE will check for valid characters when naming dialogues (a-z, A-Z, 0-9, _$#&) to prevent errors.
- Added new witch character example to example1 scene.
- Added item look up with name replacement to example1 scene.
- Optimized and improved exampleUI.cs (UI Manager example)
- Fixed connection lines not properly refreshing after connecting a node
- Fixed handling of empty NPC comments for example.UI and VIDE_Data
- Fixed windows not repainting when releasing drag outside the editor
- Fixed VIDE Editor not opening correctly after closing Unity with it opened.
- Fixed exceptions that happened when there were no dialogues created
- Reinforced error catching on VIDE_Data and VIDE_Editor
- Updated some art.
- Updated Documentation.

Note: Always remember to backup your current dialogues and modified core scripts, if you have any.

Version 1.0.3

- Fixed major bug where the scene wouldn't detect or save the changes made to the VIDE_Assign component. Thus, assigned dialogue was not being remembered and

returned errors.
Thanks to RedDeer and Greg Meach for the help.

Version 1.0.2

- Now files will be identified by their ID in the VIDE_Assign component to prevent file index offset when creating a new dialogue that affected every created dialogue.
- Dialogue folder will be refreshed now when creating and deleting dialogues.
- Added **playerTag** variable to NodeData. Now you can also assign a tag to the player node in the VIDE Editor.
- Updated Documentation.

Note: When updating to this version, please make sure that, from the VIDE Editor, you save every dialogue you have already created. This will create their corresponding IDs to prevent auto-assigning the wrong dialogue.

Version 1.0.1

- Fixed issue with titleContent to add support for Unity 5.0
- Fixed VIDE_Assign component not loading the dialogues correctly when importing the asset.
- Fixed NPC texts not properly clearing (ExampleUI.cs)
- Changed Canvas Scale Mode to constant pixel size for better consistency between aspect ratios.
- Renamed all classes/scripts to prevent duplicated definitions.
- Cleaned some scripts.
- Updated documentation

Note: If updating from the initial release, backup your Dialogues folder (VIDE/Resources/Dialogues), update, and replace the Dialogues folder. You might have to set the VIDE_Assign (DialogueAssign) components again.

Version 1.0

Initial release.