Versatile Interactive Dialogue Editor (VIDE) 2.0.2

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

Index:

What is VIDE?

VIDE (Versatile Interactive Dialogue Editor) is a plugin for Unity3D that simplifies the creation of complex, interactive dialogues by providing the user with a powerful node interface. VIDE organizes the node data from the created dialogues and presents it to the coder in a friendly, raw fashion. It is designed to be adaptable to any custom dialogue interface and communicate with it.

Create complex dialogues with the VIDE Editor, then easily incorporate and manipulate the node data to any system. You decide how you handle the data.

Some of the key features VIDE Lite includes:

- VIDE Editor: Simple, yet powerful node interface
- Add sprites, text, tags, and more to your nodes
- Use Action nodes to call your in-game methods
- Have control over the flow and start of the conversation during runtime
- Be creative! The system is not limited to conversations only
- Switch between NPC and Player-styled nodes
- Multiple comments per node
- Treat nodes as limitless, multiple-choice nodes for the player
- Extra data and Extra variables per node to add more functionality
- No components required for system to work
- Start multiple dialogues at the same time
- Compatible with most GUI and localization plugins

Extra features included in the Pro version:

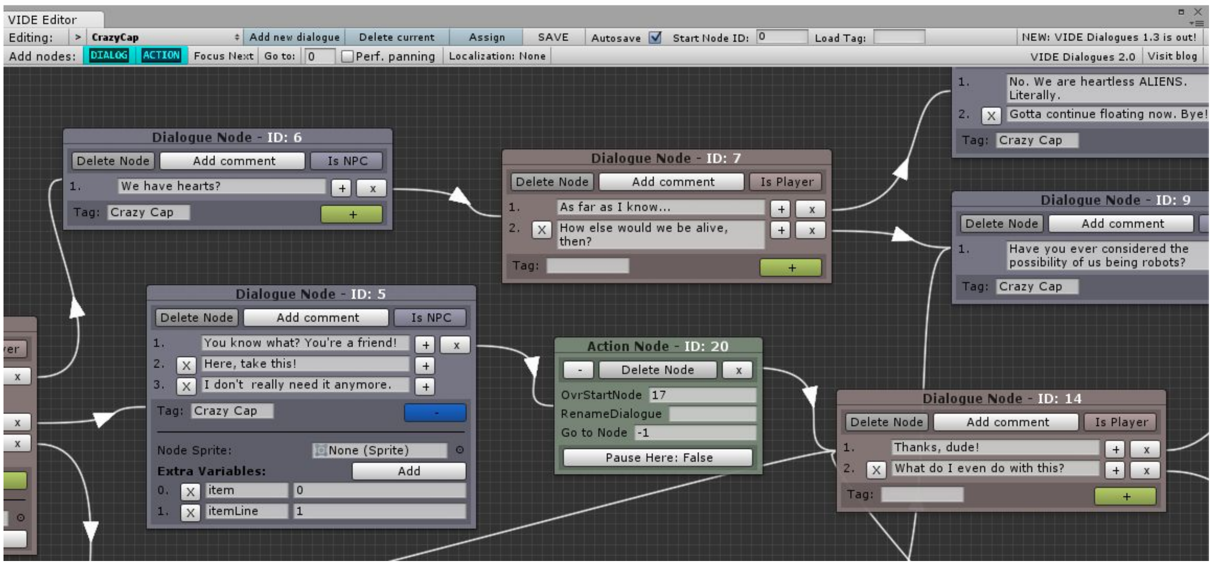
- Built-In Localization system
- VD2 class: Create multiple dialogue instances and run them at the same time.
- More demo scenes
- Per-comment audio and sprites
- Pre-defined actions for Action Nodes
- Assign View for VIDE Editor
- VD component to display Load info
- Rearrange comments in the VIDE Editor
- Select and move multiple nodes

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

What You’ll be Using

To use VIDE Dialogues, you’ll be playing around with the following:

- **VIDE Editor:** Here you can create, modify, and assign new dialogues. Accessible from Window>VIDE Editor or from a VIDE_Assign.



Dialogue Node - ID: 6
1. We have hearts?
Tag: Crazy Cap

Dialogue Node - ID: 7
1. As far as I know...
2. How else would we be alive, then?
Tag:

1. No. We are heartless ALIENS. Literally.
2. Gotta continue floating now. Bye!
Tag: Crazy Cap

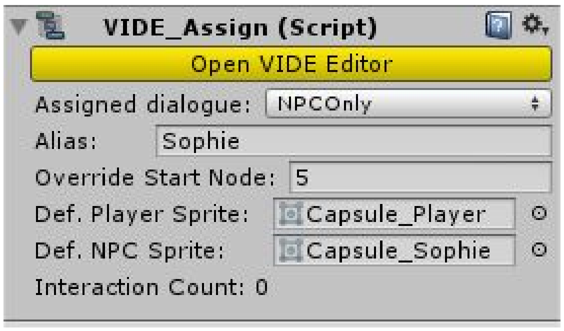
Dialogue Node - ID: 9
1. Have you ever considered the possibility of us being robots?
Tag: Crazy Cap

Dialogue Node - ID: 5
1. You know what? You're a friend!
2. Here, take this!
3. I don't really need it anymore.
Tag: Crazy Cap
Node Sprite: None (Sprite)
Extra Variables:
0. item 0
1. itemLine 1

Action Node - ID: 20
OvrStartNode 17
RenameDialogue
Go to Node -1
Pause Here: False

Dialogue Node - ID: 14
1. Thanks, dude!
2. What do I even do with this?
Tag:

- **(OPTIONAL) VIDE_Assign component:** A component that will be added to the game objects you’ll want to interact with. You assign a dialogue to it and modify some properties.

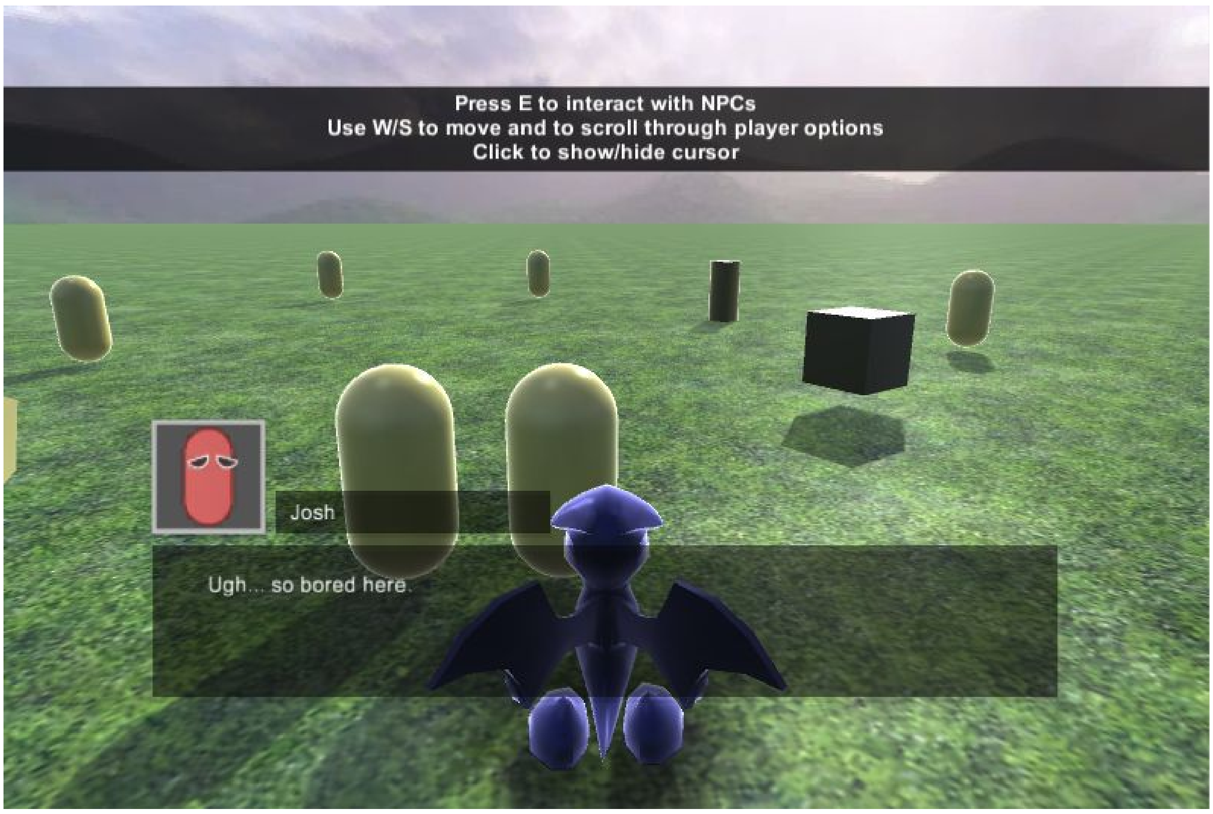

VIDE_Assign (Script)
Open VIDE Editor
Assigned dialogue: NPCOnly
Alias: Sophie
Override Start Node: 5
Def. Player Sprite: Capsule_Player
Def. NPC Sprite: Capsule_Sophie
Interaction Count: 0

- **Your favorite coding software:** Yes, you’ll still have to program a little bit.

How do I get started?

For a step-by-step guide, check the **Usage** chapter.

To get started, it is highly recommended that you first 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.


Press E to interact with NPCs
Use W/S to move and to scroll through player options
Click to show/hide cursor
Josh
Ugh... so bored here.

Example 1: Covers player movement, interaction with NPCs, starting up conversation, disabling comments, checking for and modifying Extra Variables to do special actions, NPC tags, modifying the conversation's start point, using sprites, using action nodes and their predefined actions, and updating the in-game dialogue interface using Unity's new UI system. Also has a JS version.

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

[Pro]**Example 3:** Covers Localization and a different UI manager setup that uses no components.

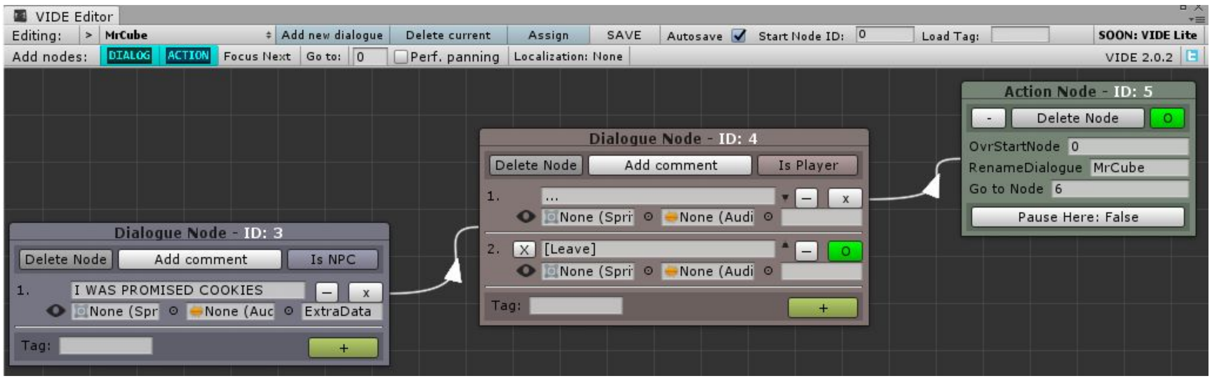
[Pro]**Example 4:** Uses the VD2 class to create multiple instances of dialogue data.
All example scripts are heavily commented so that you know what's going on. Make sure you also check the **Scripting API Reference** section of this document to get to know the variables and functions offered by the system.

While you check out the example1.scene, make sure you check out the **exampleUI** script as well (attached to the UIManager gameobject in the scene). 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 or create the dialogue you're going to modify! See next page for reference.

The VIDE Editor

This is where you will be creating your dialogues by adding **Dialogue Nodes** and connecting them. You can connect nodes as you desire. Plus, you can create Action Nodes to call your methods and to modify the flow of the conversation.



1. Toolbar

Editing: The current dialogue being edited. Click the **">"** or hit **Enter** while no node is begin highlighted to enter filtered search and choose a different dialogue to edit.

Add New Dialogue: Create a new dialogue to begin editing.

Delete current: Delete the current dialogue and load default.

[Pro]Assign: Enter Assign View to assign current dialogue to game objects.

Save: Save the current dialogue changes.

Autosave: When on, your dialogue will get saved every time the window loses focus.

Start Node ID: The first node that will be loaded when calling VD.BeginDialogue().

Load Tag: Grouping tag used when calling LoadDialogues()

News area: Clickable news area. Will open browser to the news link.

Add Nodes: Drag and drop new nodes.

Focus Next: Focus view on next node based on ID.

Go to: Focus view on node with ID.

Perf. panning: Enable/Disable performance drag.

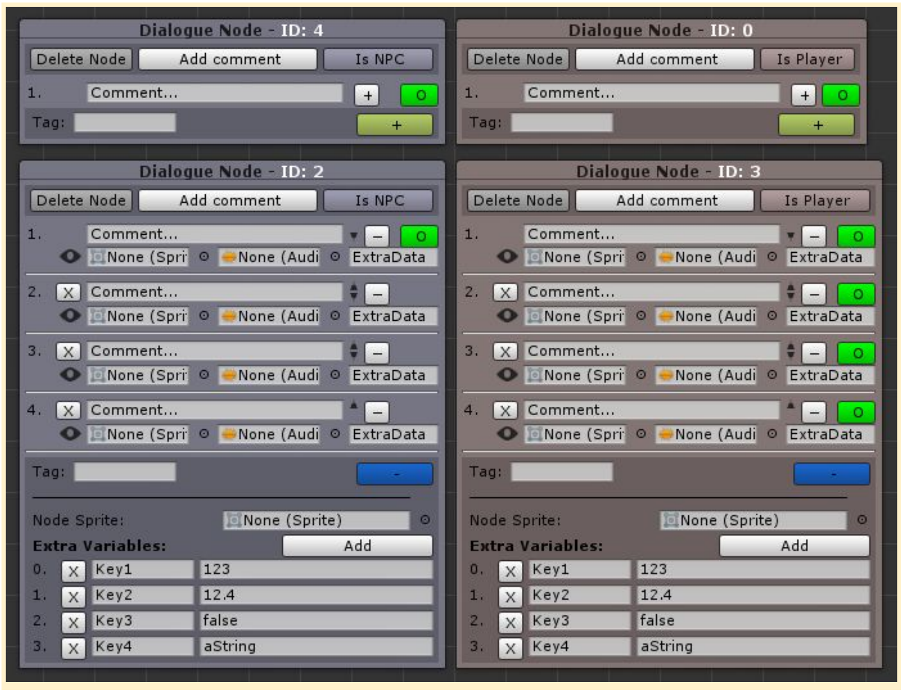
[Pro]Localization: Enter Localization menu to manage localization.

Visit Blog: Open blog on browser

2. **Dialogue Node**

Dialogue Nodes are the main source of content. They contain text, sprites, audio, tags, and extra data and variables. You can connect them to other Dialogue Nodes or Action Nodes.

Every node will have a unique ID that you can use to set the start point.

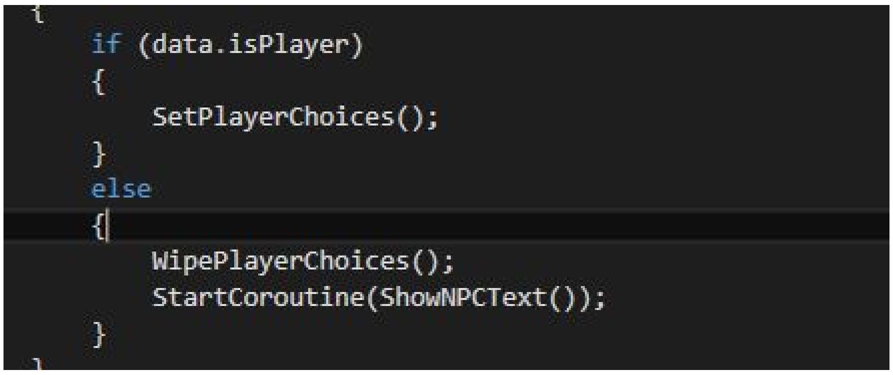


You can go simple or go complex

Format

Dialogue Nodes have two formats: **NPC** and **Player**.

NPC nodes can only connected to one node, while Player node comment can be connected to other nodes. This difference will also be reflected in the variable: **NodeData.isPlayer**. This will let you easily identify the format for you to handle the data differently.



```
    if (data.isPlayer)
    {
        SetPlayerChoices();
    }
    else
    {
        WipePlayerChoices();
        StartCoroutine(ShowNPCText());
    }
```

Comments

Each node can have an infinite amount of comments. You'll be able to retrieve these through a string array: **NodeData.comments**. You also use **NodeData.commentIndex** to navigate through the array, or set it as the player choice (See *VD.Next()* in the API for more info regarding the **commentIndex** behavior).

Per-comment data

Each comment can have a sprite, an audio file, and extraData. You can use the extra data field to pass some extra information. You can retrieve it as a string array from **NodeData.extraData**.

Sprites

You can add sprites to the default dialogue sprites, the dialogue node, and to each comment (**[Pro]**).

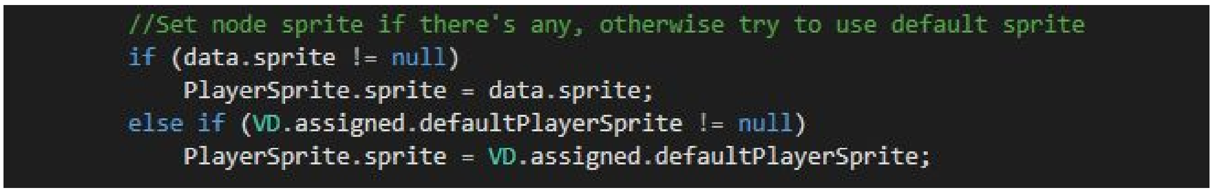
Within the Inspector, you can set the default sprites for Player and NPC nodes.
Within each node and node comment, you can set even more sprites.

Sprites (**and audios**) have to be inside a **Resources** folder to be loaded. Remember you can have as many Resources folders as you want and inside any directory you want. If you **don't** want to use the Resources folder, you can have your own array of referenced sprites and audios and pass it to *VD.spriteDatabase* and *VD.audioDatabase*. VIDE will use those arrays instead.

You can access the default sprites from **VIDE_Assign.defaultPlayerSprite** and **VIDE_Assign.defaultNPCSprite**.

The sprites will be available in NodeData as **sprite**, and **commentSprites**.

Have a look at the **NodeChangeAction** method in **exampleUI.cs** for reference on one way to handle these variables.



```
        //Set node sprite if there's any, otherwise try to use default sprite
        if (data.sprite != null)
            PlayerSprite.sprite = data.sprite;
        else if (VD.assigned.defaultPlayerSprite != null)
            PlayerSprite.sprite = VD.assigned.defaultPlayerSprite;
```

Extra Variables

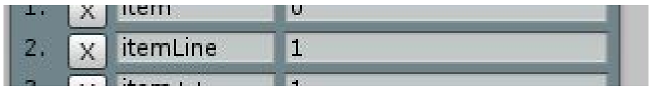
Extra Variables allows the user to store **string, int, float,** and **bool** values into a dictionary using custom keys. This info will be stored to NodeData as **extraVars**. It will grant you with more control over the behavior of your nodes.

From the VIDE Editor, you can set the key and the value for as many dictionary entries as you need.

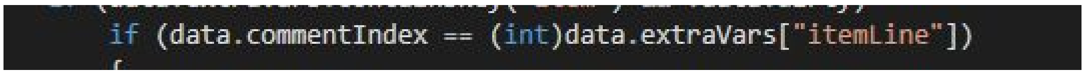
Depending on how you type the value, it will be **parsed** correspondingly into a **<string, object>** dictionary.

- Bool detection is **not** case sensitive and can have **blank spaces**.
- Float detection will accept: **, . + - e** and **blank spaces.**
- Int detection will look for a series of integral-digits.
- String will be the default when none of the above were detected.

Use the dictionary key to retrieve the value and **cast** the **object** to the correct type. If unsure of the type, remember you can **GetType()**



```
1. X item      0
2. X itemLine  1
```



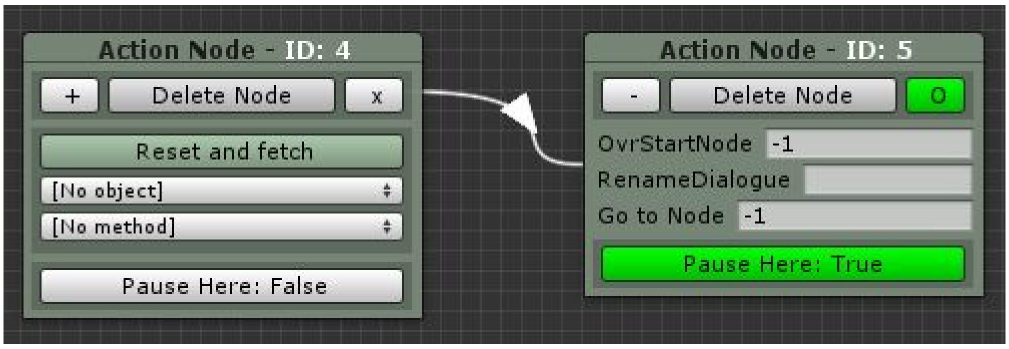
```
if (data.commentIndex == (int)data.extraVars["itemLine"])
```

Have a look at the **ExtraVariablesLookUp** method in **exampleUI.cs** for reference on how Extra Variables are being handled.

You can modify the Extra Variables during runtime! Call **VIDE_Data.SetExtraVariables()** and pass the required parameters to set the new values. You can only modify dialogues that are **loaded**, so make sure you **LoadDialogues()** first.
Check out the Scripting API for detailed information on the method.

3. **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. You can also use predefined actions.



How to use:

1. From the toolbar 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. If you desire, you can add extra predefined actions by clicking the [+] button. See below for more information.
7. Lastly, you have the option to **pause** the flow of the conversation, just like it happens when reaching a Dialogue 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. Normally, Action Nodes happen instantly and travel to the next node automatically.
8. 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 some action nodes in action!
Also talk to **MrCube** to see how predefined actions work!

Method search rules:

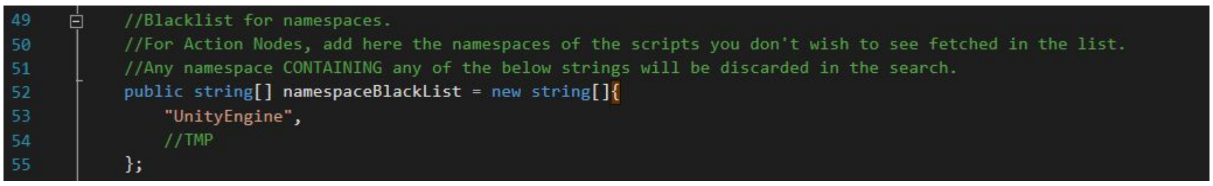
When searching for available methods, the Action Node will follow these rules:

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:



```
    //Blacklist for namespaces.
    //For Action Nodes, add here the namespaces of the scripts you don't wish to see fetched in the list.
    //Any namespace CONTAINING any of the below strings will be discarded in the search.
    public string[] namespaceBlackList = new string[]{
        "UnityEngine",
        //TMP
    };
```

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.

If you rename an object or method, you will have to Fetch and Reset again!

Callback event:

You can also subscribe to the ***OnActionNode*** event to add even more functionality. The event is called when an action node is triggered, sending the node's ID. You could forget entirely about the method-calling and the predefined actions and only use events instead, or you could use all of them to your advantage.

See the API down below for more information.

[Pro] Predefined actions:

By clicking the [+] button in the Action Node you'll be able to display the following actions:

Override Start Node: Change the start node of the conversation by modifying this variable. Use the ID of the new start node. If left on any number < 0, the action will be ignored.

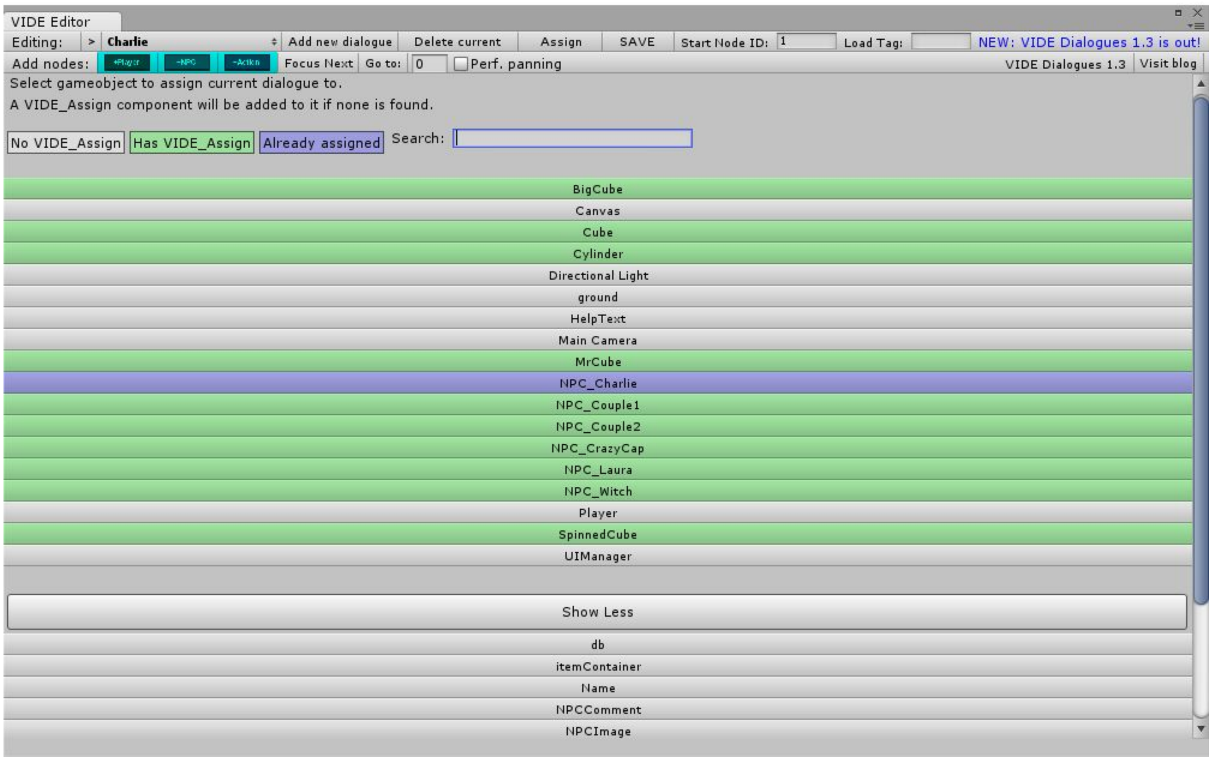
Rename dialogue: Rename the (assigned) dialogue name. If left empty, the action will be ignored.

Go to Node: Ignores the node connection and goes to specified node after executing action. If Pause Here is activated, then it will go to the node once Next() is called again on the node. Leave on **-1** to not use this action.

[Pro] Assign View

Click the Assign button to toggle Assign View. From here, you can assign the current dialogue to the game objects that you click in the list.

This is just an alternative to having to manually attach the component and select the dialogue from a dropdown list.



Select gameobject to assign current dialogue to.
A VIDE_Assign component will be added to it if none is found.

No VIDE_Assign | Has VIDE_Assign | Already assigned | Search:

BigCube
Canvas
Cube
Cylinder
Directional Light
ground
HelpText
Main Camera
MrCube
NPC_Charlie
NPC_Couple1
NPC_Couple2
NPC_CrazyCap
NPC_Laura
NPC_Witch
Player
SpinnedCube
UIManager

Show Less

db
itemContainer
Name
NPCComment
NPCImage

A VIDE_Assign component will be automatically created if none is found attached to the game object.

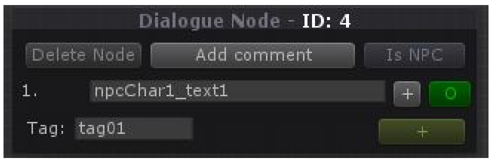
Use the Search to filter out the list.

Normal list will display a list of active/enabled objects in your scene. Only objects with HideFlags.None will appear.

Click the “Show More” button to display game objects that are disabled and/or not in the scene.

Assigning a VIDE_Assign component to a gameobject is optional, though doing it will give you much more data visibility.

[Pro] Built-in Localization



1. npcChar1_text1

Tag: tag01

VIDE is already compatible with localization systems that work with keys. Instead of inputting the actual text in the nodes, you would be typing keys instead.

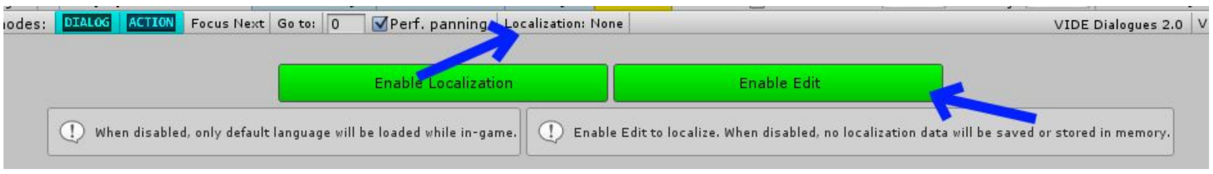
Afterwards, while in-game and before setting the text to the UI, you'd fetch the text linked to the key by using the key. Most localization plugins such as I2 have a method that returns the translation: *I2.Loc.ScriptLocalization.Get(string key).*

But as of version 2.0, VIDE includes a Built-In localization system that might be a little handy. Unlike other localizers, the difference here is that you don't need to manage keys, and you can input your localized **text**, **sprites**, or **audios** directly on the nodes without creating a new dialogue.

As of 2.0, it's the first iteration of the system. Improvements and features will of course be added in the future once I receive more feedback.

How does it work?

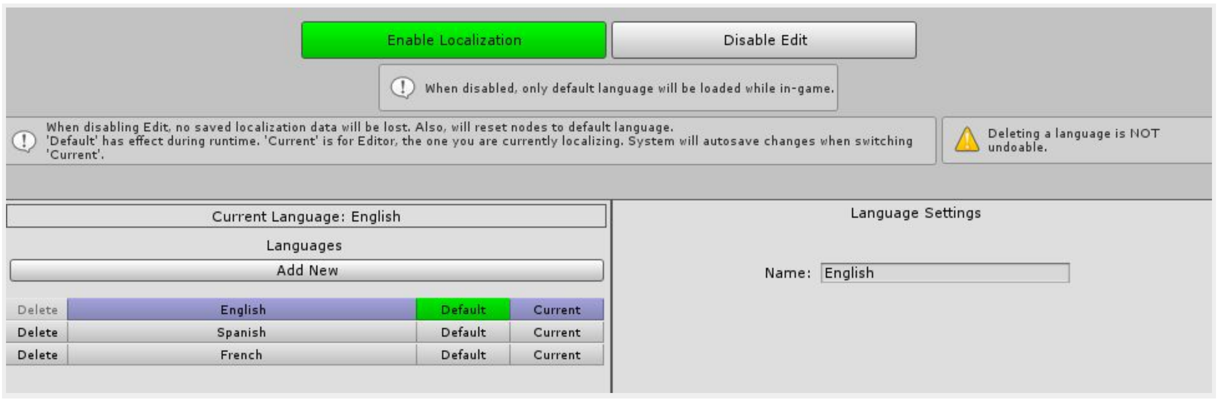
When you enable Edit from the Localization menu, you'll be able to add as many languages as necessary:



When disabled, only default language will be loaded while in-game.

Enable Edit to localize. When disabled, no localization data will be saved or stored in memory.

While Edit mode is enabled, language and localization data will be stored both in memory and in a .json file **for that dialogue.**



When disabled, only default language will be loaded while in-game.

When disabling Edit, no saved localization data will be lost. Also, will reset nodes to default language.
'Default' has effect during runtime. 'Current' is for Editor, the one you are currently localizing. System will autosave changes when switching 'Current'.

Deleting a language is NOT undoable.

Current Language: English

Delete	English	Default	Current
Delete	Spanish	Default	Current
Delete	French	Default	Current

Name: English

When you create and name a new language by clicking 'Add New', the system will be saving a copy of the localized keys for that language into a .json file located inside *VIDE/Resources/Localized*. And it will do the same for each language, all into one same file.

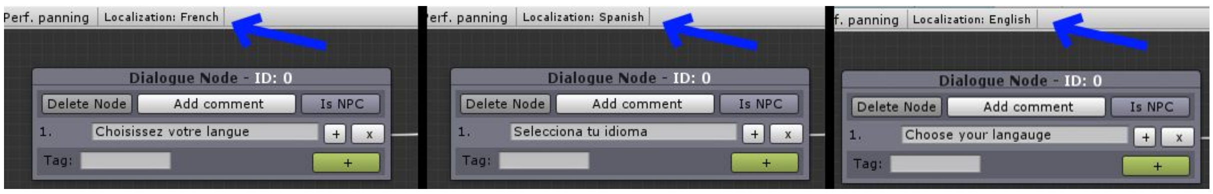
Make sure you name your languages properly!

What then?

Keys are automatically handled by the system, as the data is already bound to the node structure. All you have to do is to select the **default** and **current** languages.

Default: The default language that will loaded by **VD** during runtime. And the default language that is gonna be set when disabling Edit.
Current: The current language that you are editing in the VIDE Editor.

When you click **Current**, VIDE will first save your latest localization changes and then switch the language being currently edited. Once you do this, you can close the localization menu and edit your nodes with the localized data.



While you edit a different language, you’re editing the exact same nodes which contain the exact same Extra Variables and settings, only the **comments, tag, sprites,** and **audio** will vary between languages. You can even continue building your dialogue while in a different language, adding nodes, deleting them, etc, and the other languages will keep in sync.

- Every time you save, changes to localization will also get saved for the current language.

- When you disable Edit, the current language will be switched back to the default language. No new localization data will be recorded and saved.

- You will **not** lose any data when disabling Edit.

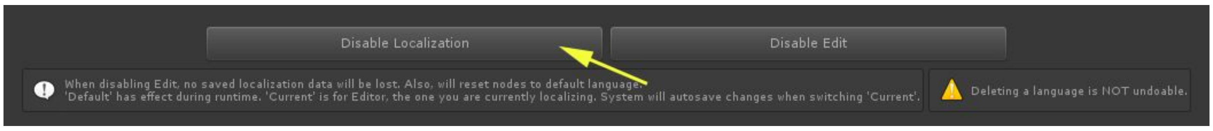
- The only way to erase localization data is by either deleting the associated dialogue or by deleting the language and then saving or switching current.

- Do NOT manually delete/modify the .json file inside the Localized folder. Same goes for the main dialogue files inside Dialogues folder.

- You can copy the localization nodes from the default language to the current language. Just click the ‘Copy Localized From Default’ in the Language Settings.

I’m done. What now?

Enable Localization so that it loads data to memory while in-game. This option will not have any effect while editing, only during runtime.



When disabling Edit, no saved localization data will be lost. Also, will reset nodes to default language.
'Default' has effect during runtime. 'Current' is for Editor, the one you are currently localizing. System will autosave changes when switching 'Current'.

Deleting a language is NOT undoable.

When you run the game and call **VD.LoadDialogues** or **VD.BeginDialogue**, dialogues will be loaded in the language set to default, which is also the **current language**.

To change the language while in-game, you call **VD.SetCurrentLanguage** and specify the name of the language. Doing so will change the **VD.currentLanguage.** Any loaded dialogues will be reloaded in the new language. Also, active VD.nodeData will be updated with the localized data.

- You can fetch the available languages by calling **VD.GetLanguages().** It will return a string array with the names.

- Use the **VD.OnLanguageChange** event to update your current dialogue, if necessary.

- For a simple demonstration of the system, check the **example3** scene included in the package along with the **UIManager2.cs**.

- For VD2 instances, you’ll have to restart the active dialogue in order for it to update the language.

- Make sure you also check the Scripting API.

If you have any issue, suggestion, or question, please don’t hesitate on contacting me.

VIDE Usage

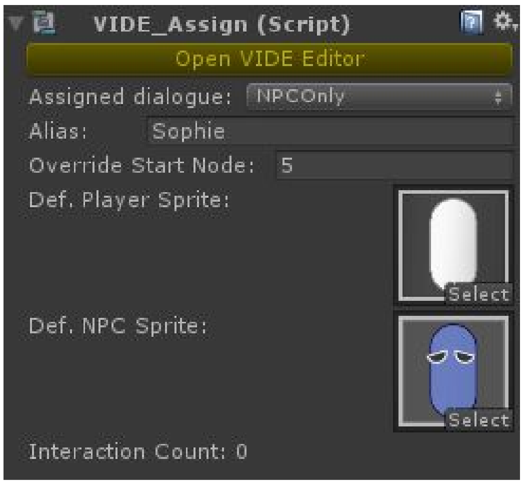
1. In Unity, go to *Window* > *VIDE Editor* to open up the editor. You can also open it from the VIDE_Assign component.
2. Click “Add new dialogue”. Name your new file and click ‘Create’. Valid characters: **a-z, A-Z, 0-9, _$#&**
3. Drag and drop a Dialogue Node from the toolbar icons.
4. Click and hold 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. Any disconnected comment or Action node will set the end of the conversation.
6. Make sure you specify the Start Node ID within the toolbar with an existing node ID.
7. Save your dialogue by clicking the ‘Save’ button.
8. Click the Assign button in the Editor. This will open the Assign View. Select the game objects you want to assign the dialogue to. Optionally, you can just attach the **VIDE_Assign** component manually and pick the assigned dialogue from the dropdown in the Inspector.
9. Optionally, you can setup an alias, overrideStartNode, and default sprites for the **VIDE_Assign** component.
10. Have a gameobject with a **VIDE_Data** component attached. VIDE_Data is in charge of handling all of the dialogues during runtime.
11. Lastly, have or add a script that will manage all UI-related stuff in your desired game. object. Remember you can use the **exampleUI** script that comes with this asset as a startpoint. Check the **How it works** chapter for a more detailed explanation.

Tips and good-to-knows:

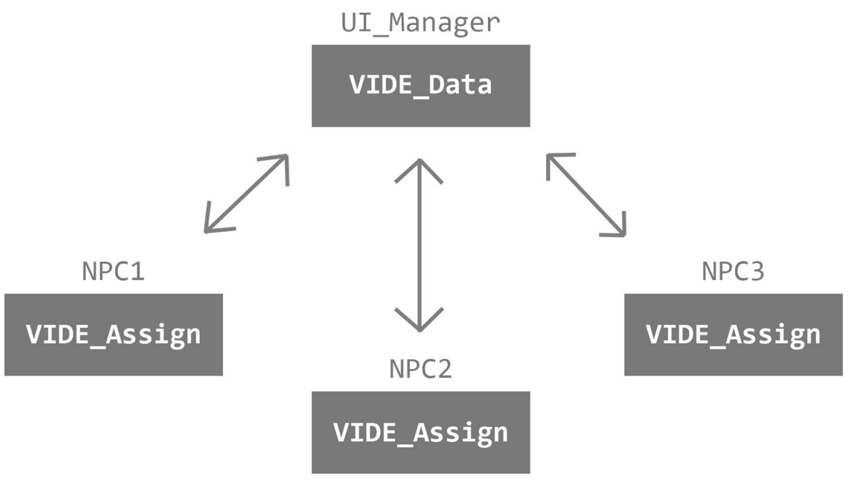
- Name your dialogues under certain nomenclature to easily filter them out in Dialogue Search.
- Hit **Enter** while no node is begin highlighted to enter Dialogue Search and choose a different dialogue to edit. Useful when you have a lot of dialogues.
- Hit **Esc** to exit from Assign View or Dialogue Search.
- Duplicate nodes by **right-clicking** and **drag&dropping**.
- Hold **Shift** while clicking nodes to select multiple of them and then drag them together.
- Release a connection line on empty space to create a new node and connect automatically.
- 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.

How it works

Initially, you'll require only your dialogues and some code to get things working, though you are encouraged to use the VIDE_Assign component as it gives you greater access and visibility.


VIDE_Assign (Script)
Open VIDE Editor
Assigned dialogue: NPCOnly
Alias: Sophie
Override Start Node: 5
Def. Player Sprite:
Def. NPC Sprite:
Interaction Count: 0

When you start a conversation with **VD.BeginDialogue()**, you send the **VIDE_Assign** to the **VD**. **VD**'s methods and variables are **public static** and can be accessed/called from everywhere. You can also start a conversation by specifying a name. In that case, a temporal, hidden VIDE_Assign will be automatically created and linked to the dialogue.



The **VD** component contains a variable called **nodeData**. This variable of type **NodeData** stores, as you might imagine, 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 new current node. Note that Action nodes will not modify nodeData.
Essentially, you're constantly reading the contents of *VD.nodeData* to do whatever you want in your dialogue interface using its data. You can achieve this in a better way by subscribing to the available events like **OnNodeChange**. You might also want to use the VIDE_Assign variables. During an active conversation at runtime, access the VIDE_Assign component through **VD.assigned**.

Basic coding workflow:

1. User calls ***VD.BeginDialogue()*** method on **VD** to begin the conversation with the NPC. The method requires the user to send the **VIDE_Assign** component attached to a game object **or** a dialogue name. This will populate the ***VD.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. You can do this through your own methods or by using the callback events.

3. User calls ***VD.Next()*** on **VD** to populate *VD*.nodeData with the data from the next Node in the conversation. For Player nodes, the system will read the **nodeData.commentIndex** variable to know where to go.

4. User uses the new data in *VD.nodeData* to customize the in-game dialogue interface.

5. Step 3 & 4 repeats until user reads the ***nodeData.isEnd*** variable or ***VD.OnEnd*** event to know when to call the ***VD.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.

Loading/Unloading dialogues

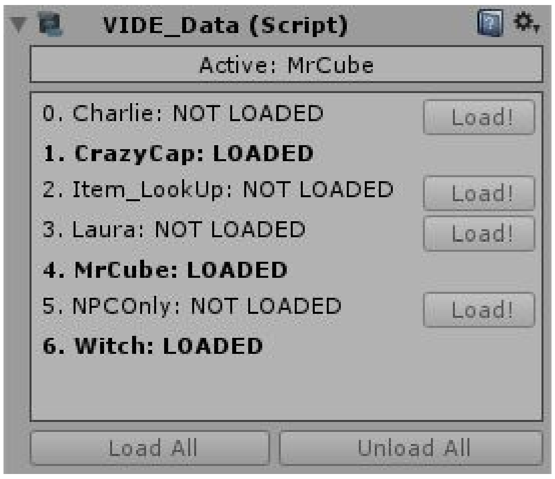
When you **BeginDialogue()**, **VD** will attempt to load the dialogue data from disk **if** it isn't already loaded.

As the project grows, you might find yourself with hundreds of dialogues with dozens of nodes each. When you see a delay when starting a conversation, it is probably because there's already a lot of data to be loaded. In this case, it would be better if you loaded the dialogue beforehand (e.g. when loading the scene), so that when you interact with the NPC, the dialogue will be already loaded, posing no delay.

You can do this by calling **LoadDialogues()** and **UnloadDialogues()** methods on **VD**. The dialogues will remain loaded on the component. If it gets destroyed, you'll have to load the dialogues again. See the Scripting API down below for more information on the methods.

You only call these methods on VD. VD2 will grab dialogue data from VD.

[Pro] **VD** has a custom editor in the Inspector. It will display information about the currently active dialogue (if there's any) and the loaded and unloaded dialogues. You can also quickly load all or unload all by clicking the corresponding buttons. Of course, this won't be of any use outside the Editor. **You will have to rename 'VIDE_Data' component to 'VD' in order for it to work.**


VIDE_Data (Script)

Active: MrCube

0. Charlie: NOT LOADED
1. CrazyCap: LOADED
2. Item_LookUp: NOT LOADED
3. Laura: NOT LOADED
4. MrCube: LOADED
5. NPCOnly: NOT LOADED
6. Witch: LOADED

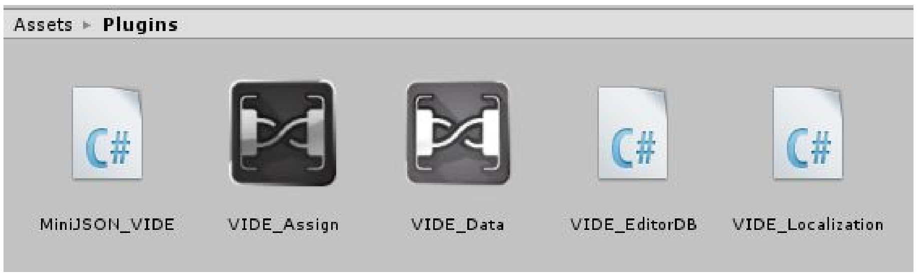
*Note: Dialogues are **not** unloaded when ending a conversation.*

Javascript example scene

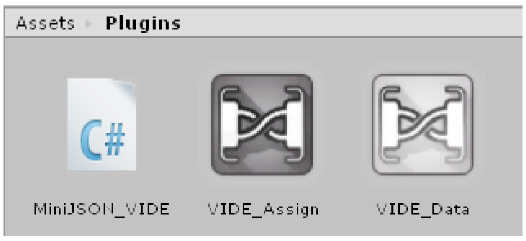
JS is fully compatible with VIDE. The plugin includes an example scene that uses a UI manager written in JS.
The scripts are commented out as they would not properly compile for the plugin's default environment which is C#. To use the ***JS_example1*** scene and access the plugin's classes from JS, please do the following:

Create or have a folder named **Plugins** in the root of the project.

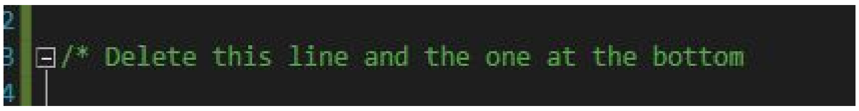
For the **Pro** version, move these scripts to the Plugins folder:



For the **Lite** version, move these scripts to the Plugins folder:

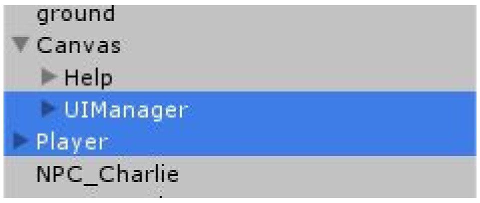


Now you are free to uncomment **JS_exampleUI.js** and **JS_demoPlayer.js**. Then save both scripts and let Unity recompile.



```
/* Delete this line and the one at the bottom
```

Before hitting play in the scene, make sure the scripts are correctly attached to the 2 corresponding gameobjects:

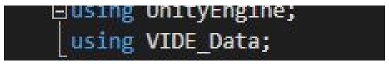


Differences with C# version

1. Example1 scene has a Quest Chart demo. The script is in C#, but it works almost exactly the same way for JS.
2. “Almost” because to unbox the extraVars values, you call VD *Get* methods instead of typecasting the variables. See JS_exampleUI.js for reference.
3. Unlike the C# version, the JS version automatically finds the references to the objects and components in the scene. It saves you the pain of setting things up in the Inspector, which you would have to when uncommenting the scripts.

Scripting API reference

***namespace* VIDE_Data**
Remember to import this to easily access the VD class.



```
using UnityEngine;
using VIDE_Data;
```

public class **VD : MonoBehaviour**

This class will store and manage the dialogues at runtime. As an optional component, it will display dialogue info on the Inspector. (Rename "VIDE_Data" script to "VD" in order to use as component)

public class **VD2 [Pro]**

This is the same same as VD, but it has no static members. Unlike VD, you can use it to create multiple instances of it and manage multiple dialogues at the same time. Some members are VD-exclusive, like the Loading/Localization-related methods. If using VD2, you can still access those through VD.

For both classes, you can call and access the following variables, methods, and events. Note static members only apply to VD.

Functions:

public static NodeData **BeginDialogue(**VIDE_Assign diagToLoad**);**
Initiates the dialogue just sent. Tries to load it from disk if not loaded. 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 Dialogue Node.

(Overload) public static NodeData **BeginDialogue(**string diagToLoad**);**
Initiate the dialogue specified by name. A temp VIDE_Assign will be linked to the dialogue while it remains loaded. It will have **default** values. Access it during an active dialogue through the **assigned** variable. Modify the same variables you modify from the Inspector by calling **SetAssigned** method.

public static string[] **GetDialogues();**
VD-only. Returns the list of created dialogues.

public static void EndDialogue();
Resets all temporal data. Raises VIDE_Assign's **interactionCount** before unlinking it. Do not call BeginDialogue() again if you haven't called this yet. This will not unload the dialogue from memory.

public static 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.commentIndex** is correctly set before calling Next(). If current nodeData belongs to an NPC Node with multiple comments, calling Next() will advance through those comments before getting to the next node.
Will trigger **OnNodeChange** event.
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(), or listen to the OnEnd event.
Also returns the current **NodeData** package.

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

public static NodeData GetNext(bool returnNextComment, bool callAction);
Simulates Next() method. Returns next node's NodeData package based on current nodeData. Will not fire any events. Will not modify current nodeData. Will return current nodeData if next is null or if it is an ActionNode.

public static NodeData GetNodeData(int ID);
Returns the NodeData package of the specified node in the active dialogue. This doesn't affect the current progression of the dialogue. nodeData will remain the same.

public static NodeData GetNodeData(string dialogueData, int ID, bool forceLoad);
Returns the NodeData package of the specified node in the specified dialogue. You can use it with inactive dialogues that are **loaded.** If they are not loaded and you still want the NodeData, make sure you set forceLoad to true.

public static string GetFirstTag(bool searchPlayer);
Returns the first tag it finds. Does not follow node structure. If searchPlayer is false, then NPC nodes will be searched.

public static void **LoadDialogues();**
VD-only. Loads all of the dialogues to memory.

(Overload) public static void **LoadDialogues(**string dialogueName, string loadtag**);**
VD-only. Loads the desired dialogue(s) to memory. Leave dialogue name empty to load all of the dialogues tagged as *loadtag*. Leave *loadtag* empty to load the specified dialogue. Use both fields to load a dialogue under a specific tag.
You can set the dialogue's loadTag in the toolbar within the VIDE Editor.

public static void **UnloadDialogues();**
VD-only. Unloads all of the dialogues from memory.

public static void **SetExtraVariables**(string dialogueName, int nodeID, Dictionary<string, object> newVars**);**
Updates the Extra Variables of the desired dialogue and node with new content. It modifies a **loaded** dialogue, not NodeData, so the changes will persist while the dialogue remains loaded. If successful and if the dialogue is currently active, sets NodeData.dirty to true. Refer to exampleUI.cs for a demo.

(Overload) public static void **SetExtraVariables**(int nodeID, Dictionary<string, object> newVars**);**
Updates the **active** dialogue node's Extra Variables. If successful, sets NodeData.dirty to true. Refer to exampleUI.cs for a demo.

public static Dictionary<string, object> **GetExtraVariables**(string dialogueName, int nodeID)
Get a Dictionary with the Extra Variables of a node in a loaded dialogue. Useful for spying on dialogues that are not currently active.

(Overload) public static Dictionary<string, object> **GetExtraVariables**(int nodeID)
Get a Dictionary with the Extra Variables of a node within the currently active dialogue.

public static void **SetComment**(string dialogueName, int nodeID, int commentIndex, string newComment)
Update a node's comment with a new comment. This will overwrite the localized value.

public static VIDE_Assign **GetAssigned**(string dialogueName)
Gets the dialogue's temp VIDE_Assign (the non-component).

public static void **SetAssigned**(string dialogueName, string alias, int ovr, Sprite playerSprite, Sprite npcSprite)
Sets the temp VIDE_Assign's variables for the given dialogue.

public static void **SetVisible**(string dialogueName, int nodeID, int commentIndex, bool visible)
Sets the visibility of a node comment. Dialogue doesn't have to be active. When a comment is invisible, it will not be included in Nodedata.

(Overload) public static void **SetVisible**(int nodeID, int commentIndex, bool visible)
Sets the visibility of a node comment in the active dialogue. When a comment is invisible, it will not be included in Nodedata.

public static void **SetCurrentLanguage**(string lang)
VD-only. Switch the current language. Will automatically reload any loaded dialogues. Make sure the language name exists.

public static string[] **GetLanguages**()
VD-only. Get a list of the available languages created. Add them from the Localization menu inside the VIDE Editor.

Events:

public static ActionEvent **OnActionNode;**
Called when an action node is triggered. Sends ID of the node. Subscribe methods that have one int parameter.

public static LoadUnload **OnLoaded;**
Called when finished loading dialogue(s).

public static LoadUnload **OnUnloaded;**
Called when finished unloading dialogues.

public static NodeChange **OnNodeChange;**
Called on each node change (only Player and NPC). Refer to OnActionNode event for Action Nodes.

public static NodeChange **OnEnd;**
Called when we tried to call Next on a disconnected node/comment.

public static NodeChange **OnLanguageChange;**
Called when finished setting current language.

Note: OnEnd and OnNodeChange events for the VD2 class will return the VD2 object instead of NodeData. This is useful to find the index of the instance within a list.

Variables:

public static bool **isLoaded;**
Is there any active dialogue?

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

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

public static NodeData **nodeData;**
Variable containing all of the current Node data you'll need to set up your dialogue interface. Variable is null when no dialogue is currently loaded.

public static Sprite[] **spriteDatabase;**
VD-only. Pass an array of sprites to this variable to remove the need of the Resources folder.

public static AudioClip[] **audioDatabase;**
VD-only. Pass an array of sprites to this variable to remove the need of the Resources folder.

public static bool **localizationEnabled;** (ReadOnly)
VD-only. Is localization enabled? You can set this from the localization menu in the VIDE Editor.

public static string **currentLanguage;** (ReadOnly)
VD-only. The current language set. Set it by calling SetCurrentLanguage().

public static string **defaultLanguage;** (ReadOnly)
VD-only. Returns the name of the default language. You can set this from the localization menu in the VIDE Editor.

NodeData class

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

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

public bool **isPlayer;**
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[] **comments;**
An array of strings with all of the node's comments.

public string[] **extraData;**
Per-comment extra data. Use along with commentIndex.

public Sprite[] **sprites;**
Per-comment sprite. Use along with commentIndex.

public AudioClip[] **audios;**
Per-comment audio. Use along with commentIndex.

public int **commentIndex;**
Starts always on 0. For NPC nodes, it increases with each Next() until reaching the last comment. For Player nodes, it acts as the picked choice. Manually set this variable before calling Next() on Player nodes.

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

public Sprite **sprite;**
The sprite set for this node in the VIDE Editor.

public Dictionary<string, object> **extraVars;**
The Extra Variables set in the VIDE Editor.

public bool **dirty;**
Becomes true when you modify an active dialogue's Extra Variables. It will be false again on next interaction.

VIDE_Assign

Class holding interaction information. Per-gameobject or per-dialogue component, depending on how you use it. If you don't use it as a component, you can still get access to it through GetAssigned and configure it through SetAssigned.

Functions:

public string **GetAssigned();**

Returns a string with the name of the currently assigned dialogue.

public bool **AssignNew(**string dialogueName**);**

Assign a different dialogue to this VIDE_Assign (Only as component). 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.

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 **alias;**

The custom name for this dialogue. Can be set from the Inspector. (Previously known as dialogueName)

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.

public Sprite **defaultNPCSprite;**

Default sprite assigned for the NPC node.

public Sprite **defaultPlayerSprite;**

Default sprite assigned for the Playernode.

Changelog

Version 2.0.2

- RENAMED UpdateComment and UpdateExtraVariables to SetComment and SetExtraVariables.
- Rearrange comments in the VIDE Editor
- Select and move multiple nodes
- Fixed GetNodeData(string, int, bool) nullifying currentPlayerStep during a dialogue, which could return errors.
- New comment visibility flag. Set it from the inspector and through SetVisible method.
- New spriteDatabase and audioDatabase variables to remove need of Resources folder.
- Added JS example scene.
- Added a Quest Chart demo object to example1 scene.
- Localization: fixed exceptions with new languages.
- Localization: Added 'Copy Localization from Default'. Copy localized node data from default language to current.
- Updated doc

Version 2.0.1

- Null check for method fetch. Action Nodes would return exceptions when clicking 'Reset and Fetch' while you had missing MonoBehaviors attached to game objects.
- Fixed inability to dock VIDE Editor
- You can now safely add sub-folders inside the Dialogues folder to organize your dialogues.
- Removed autofocus
- Localization & Loading/Unloading variables and methods removed from VD2. Use VD to handle those.
- VD2 instance will no longer load dialogues to memory separately. It is now synced.
- Added GetNodeData overload to access node data from inactive dialogues
- Renamed isLoaded to isActive.
- Fixed node duplication referencing Extra Variables
- Example scene 1 art overhaul

Version 2.0

- First version of built-in localization.

- Removed NPC nodes. Now it's just Dialogue nodes and Action nodes. You can toggle between NPC and Player formats for the Dialogue Node.
- VIDE_Data now a namespace to access VD and VD2 class.
- You can now use VD2 class to create multiple instances and manage multiple dialogues at the same time.
- VIDE_Data component in the scene is now optional.
- VIDE_Assign component in the scene is now optional.
- Fixed a lot of issues with the Undo system. You can now safely undo dialogue deleting and loading.
- Per-comment Audio and Sprite slots.
- You can now duplicate nodes by right clicking them.
- Added grid and snapping to Editor
- Moving a node will now mark the dialogue as dirty.
- Added “Go to Node” predefined action to Action Node
- Added ‘SetComment’ method to VIDE_Data
- Added ‘GetExtraVariables’ overload to VIDE_Data
- Added 'GetNodeData' methods to VD.
- Added 'BeginDialogues' overload.
- Added 'GetLanguages' and 'SetCurrentLanguage' methods.
- Added 'GetAssigned' and 'SetAssigned' to manage non-component VIDE_Assign.
- Added another demo character to the example1 scene.
- Added example3 scene which covers a new UI Manager and localization.
- Changed Editor view mode for better performance. Canvas is no longer infinite.
- Fixed duplicate ID bug when manually duplicating the .json dialogue files.
- Improved the node view focusing.
- Autofocus on startNode when loading dialogue.
- Fixed some unexpected behaviors when using SetNode method.
- Fixed some unexpected behaviors when using Action Node.
- Fixed exampleUI.cs allowing you to scroll through comments on NPC nodes and thus returning exceptions.
- Added News Checker to the toolbar.
- Added Dialogue Search to Editor.
- Added Assign View menu to the Editor. Will let you easily inspect and assign dialogues to gameobjects.
- Fixed undraggable Editor after releasing node outside window.
- Polished VIDE_Assign Inspector refresh.
- PathToVide deprecated. VIDE will now automatically detect new folder location in project.
- Changes to NodeData variables.
- Minor fixes and polishing.
- Updated Documentation.

Version 1.2

- Fixed VIDE_Data loading wrong dialogues (with an offset) after creating or deleting new dialogues.
- New toolbar that replaces Editor Tools.
- Add sprites to your dialogues and nodes! Select default Player/NPC sprites from VIDE_Assign component and/or set a specific one for each node. Access the sprites through VIDE_Data.assigned.defaultPlayerSprite and NodeData.nodeSprite. Sprites should go inside Resources.
- Brand new Extra Variables system. Store strings, ints, floats, or booleans as values. Access the objects from the extraVars dictionary using custom keys. Deprecated extraData is still available, though it will be removed in a later version.
- Added SetExtraVariables method to modify Extra Variables at runtime.
- Added 'extraVars', 'nodeSprite', and 'dirty' variables to NodeData.
- Added buttons to expand/collapse window content to free space.
- Added checkbox to disable the performance view when panning inside the VIDE Editor (Empty nodes when panning).
- Inspector multi-object editing is now supported for VIDE_Assign component.
- Updated exampleUI.cs and example1 scene with sprites and extraVars.
- Updated documentation.

- VIDE_Data's methods and variables are now static and can be accessed from anywhere; instance no longer needed. You'll have to update your scripts.
- Renamed VIDE_Assign.dialogueName variable to VIDE_Assign.alias. You might have to set the names again for each VIDE_Assign and update your scripts. This was a necessary change as 'dialogueName' was misleading.

Version 1.1.3f2

- Added OnNodeChange and OnEnd events to VIDE_Data
- Completely refactored exampleUI.cs. It is now a better start point for new users
- Fixed VIDE Editor not automatically opening newly-created dialogue
- Fixed a case where VIDE Editor would not save when closing or losing focus
- Fixed Action Nodes for methods that have no parameters
- Fixed new dialogues not having a default start node of 0
- Optimized nodes that are outside the visible canvas to improve VIDE Editor performance
- Now you can threaten Charlie with the Mystical Rocket-Launcher (example1.scene)

Version 1.1.3

- Now you can Load and Unload dialogues to memory whenever you want by calling LoadDialogues() and UnloadDialogues().
- Added OnActionNode event to Action Nodes.
- Added predefined actions to Action Nodes.
- Updated VIDE_Data's Inspector content.
- Added *Load Tag* to Editor Tools within the VIDE Editor.

- Fixed VIDE Editor not sorting list after creating a new dialogue.
- Minor fixes and improvements.
- Updated example1.scene and Documentation.

Version 1.1.2

- (Linux) Now clicking “Open VIDE Editor” button will open the right dialogue.
- (Linux) VIDE Editor will no longer open at fullscreen everytime.

Version 1.1.1

- Empty player extraData will now also be added to the array to keep matching index.
- Save system extra protection against errors that happen when no dialogues are found.
- Corrected "VIDE/Resources/dialogues" to "VIDE/Resources/Dialogues" for Linux.
- Fixed exampleUI.cs reading previous extraData while on a paused action node.
- Now the Editor will remember the last dialogue you were editing when opening from Window/VIDE Editor.

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.