Versatile Interactive Dialogue Editor (VIDE)
1.2
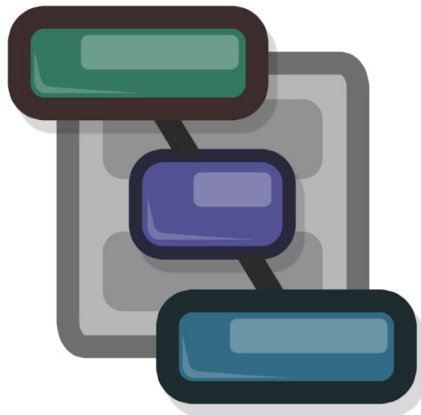
Documentation Changelog:

Version	Changes
1.2	- Added Extra Variables section - Added info on Sprites - Updated info on VIDE_Data - Updated info on VIDE_Assign - Updated Images - Updated API

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] Add sprites to dialogues and specific nodes.
- 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 Variables to your Player and NPC nodes.
- Split an NPC's comment into chunks in a single node by using the <br> 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 and modifying Extra Variables to do special actions, NPC tags, modifying the conversation's start point, using sprites, using action nodes, 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.

While you check out the example1.scene, make sure you check out the **exampleUI** script as well. 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.
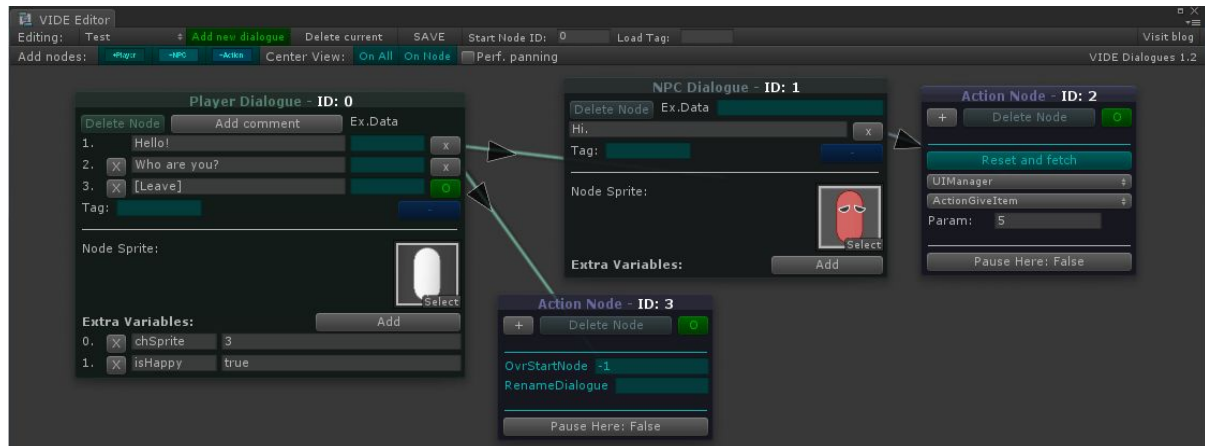
The editor consists of four main window types:

1) **Toolbar:** 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. Also, you can set the dialogue's Load Tag. [New] Disable/Enable performance panning.

2) **NPC comment node:** This node contains an NPC's comment. Add a tag, node sprite, and extra variables to store in NodeData.

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.

Sprites

VIDE 1.2 allows you to add sprites to both the dialogues and the dialogue's nodes.

Within the Inspector, you can set the default sprites for Player and NPC nodes.
Within each node, you can set a specific sprite for them.

Sprites 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.

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

The node sprite will be available in NodeData as **nodeSprite**.

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.nodeSprite != null)
    PlayerSprite.sprite = data.nodeSprite;
else if (VIDE_Data.assigned.defaultPlayerSprite != null)
    PlayerSprite.sprite = VIDE_Data.assigned.defaultPlayerSprite;
```

Extra Variables

VIDE 1.2 introduces Extra Variables, a feature that replaces extraData. Extra Variables allows the user to store **string, int, float,** and **bool** values into a dictionary using custom keys. This will give you easier and more powerful control over what happens with your nodes.
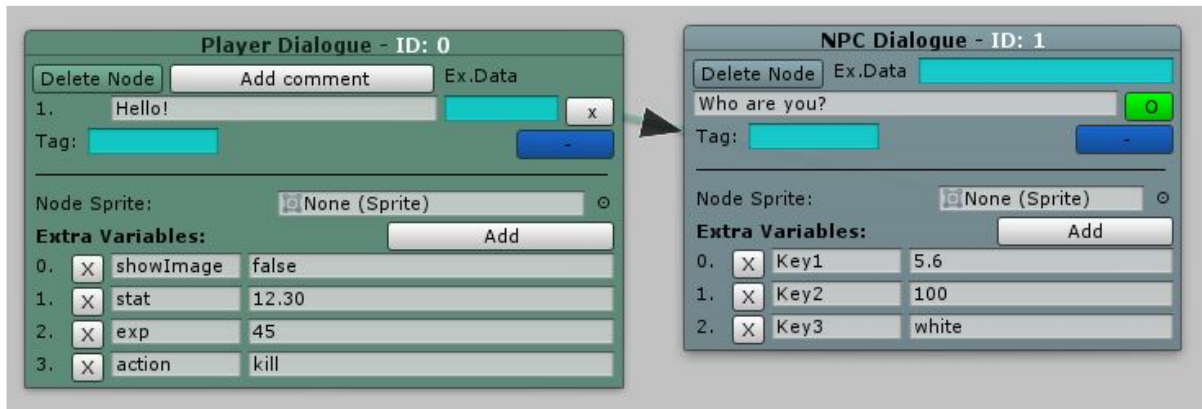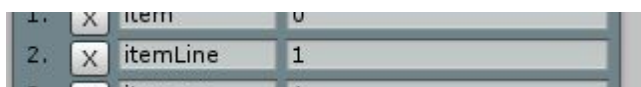

From the VIDE Editor, you can set the key and the value for as many dictionary entries as you need.

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()**

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.UpdateExtraVariables()** 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.

ExtraData variable is still available to use, but will be completely removed on a later version.

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. 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 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.
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 action nodes in action!
Also talk to MrCube to see how predefined actions work!

Tip: For now, you can connect Player nodes to Player nodes by having an Action Node (It can be empty) in-between.

Predefined actions:

VIDE 1.1.3 introduces 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.

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:

```
49      //Blacklist for namespaces.
50      //For Action Nodes, add here the namespaces of the scripts you don't wish to see fetched in the list.
51      //Any namespace CONTAINING any of the below strings will be discarded in the search.
52      public string[] namespaceBlackList = new string[]{
53          "UnityEngine",
54          //TMP
55      };
```

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.

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.

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 an alias for the dialogue.
10. Have an 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. 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**: The VIDE Editor will save whenever it loses focus (clicking outside, closing it or closing 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.
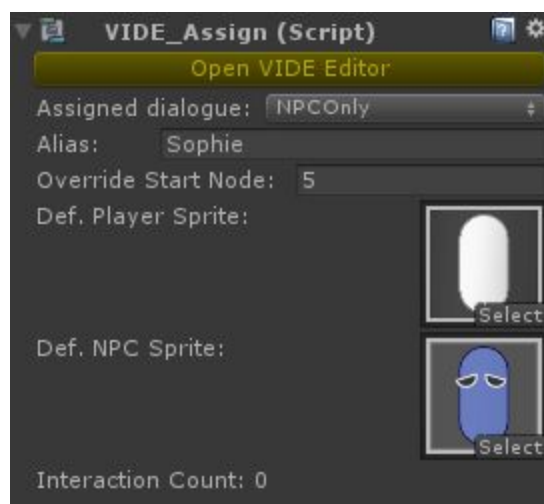
By design, you should have only **one** VIDE_Data in the scene that will act as a dialogue manager. It will keep loaded dialogues in memory and provide your UI Manager with all of the data in needs. When you start a conversation, you send the VIDE_Assign to the VIDE_Data. VIDE_Data's methods and variables are **public static** and can be accessed/called from everywhere.
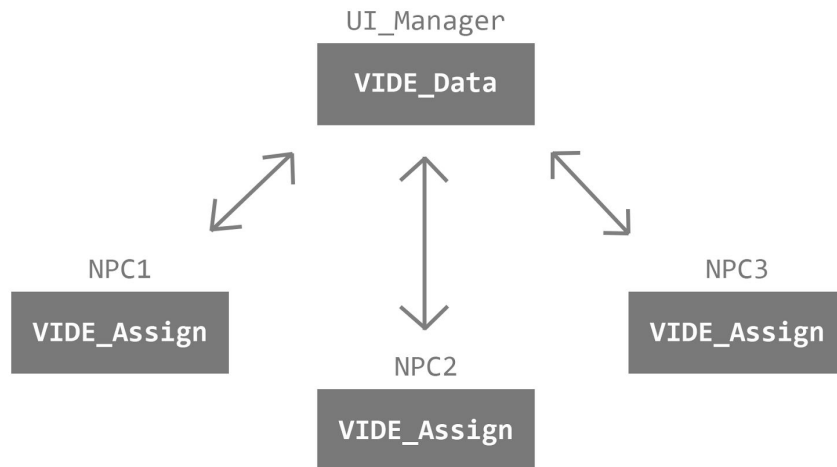
The VIDE_Data component contains a variable called **nodeData**. This variable of type **NodeData** stores, as you might imagine, all of the current node's data. At the beginning of the conversation, the **current node** is equal to the Start Node you selected in the VIDE Editor (unless you set the Start 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 *VIDE_Data.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**.

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 or *VIDE_Data.OnEnd* event 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.

Loading/Unloading dialogues

When you interact with an NPC, VIDE_Data 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 with an NPC, 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 **VIDE_Data**. 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.

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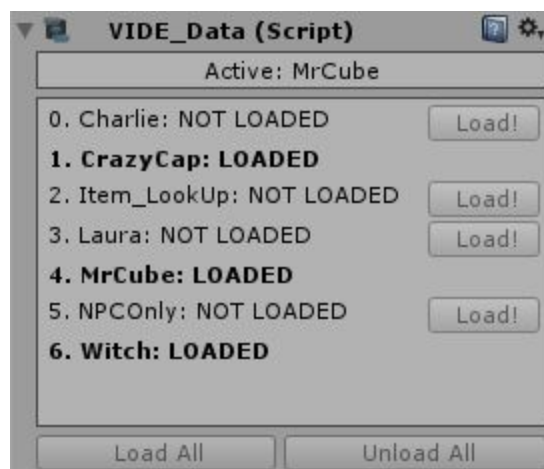
Note: Dialogues are not unloaded when ending a conversation.

Scripting API reference

VIDE_Data
Script component attached to a GameObject in your scene. This component will store all the data regarding the currently loaded dialogue. You can call and access the following static variables and methods:

Functions:

public static NodeData **BeginDialogue(**VIDE_Assign diagToLoad**);**
Activates 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 Player or NPC node.

public static 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 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.selectedOption** is correctly set before calling Next(). If current nodeData belongs to an NPC Node with multiple comments (when using <br>), 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 static 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.

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

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

(Overload) public static void **LoadDialogues(**string dialogueName, string loadtag**);**
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 from the Editor Tools window in the VIDE Editor.

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

public static void **UpdateExtraVariables(**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 **UpdateExtraVariables(**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.

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.

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's Editor Tools window. Returns 0 if there's no dialogue currently loaded.

public static VIDE_Assign assigned;
Reference to the currently loaded VIDE_Assign component. 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.

NodeData class
This class stores all of the 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 <br>, 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 <br>.

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;** [Deprecated]
The string of extra data declared in the VIDE Editor for NPC nodes. This variable will be removed in a later version. Use Extra Variables instead.

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

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

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

public Sprite **nodeSprite;**
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

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.

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.

Changelog

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 UpdateExtraVariables 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.