About

absent-variables is a completely free and open-source dialogue system designed specifically for Unity. If you want to contribute, please go to Contributing section. I'd be really happy to see you helping me with this!

In this documentation, you will learn how to use, modify and extend the system.

You can start learning by reading Installing or Basic Setup.

Edit this page

About

absent-variables is a completely free and open-source dialogue system designed specifically for Unity. If you want to contribute, please go to Contributing section. I'd be really happy to see you helping me with this!

In this documentation, you will learn how to use, modify and extend the system.

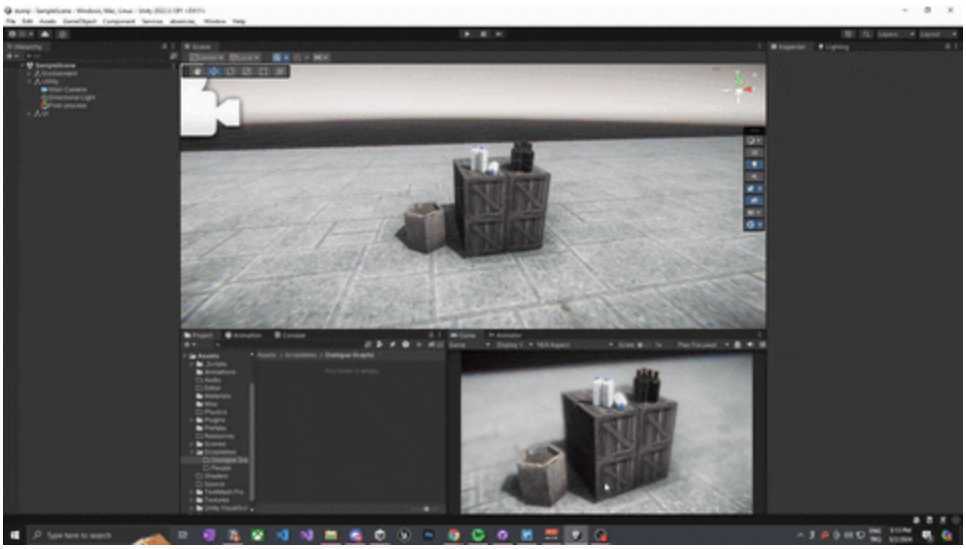
You can start learning by reading Installing or Basic Setup.

Edit this page

Basic Setup

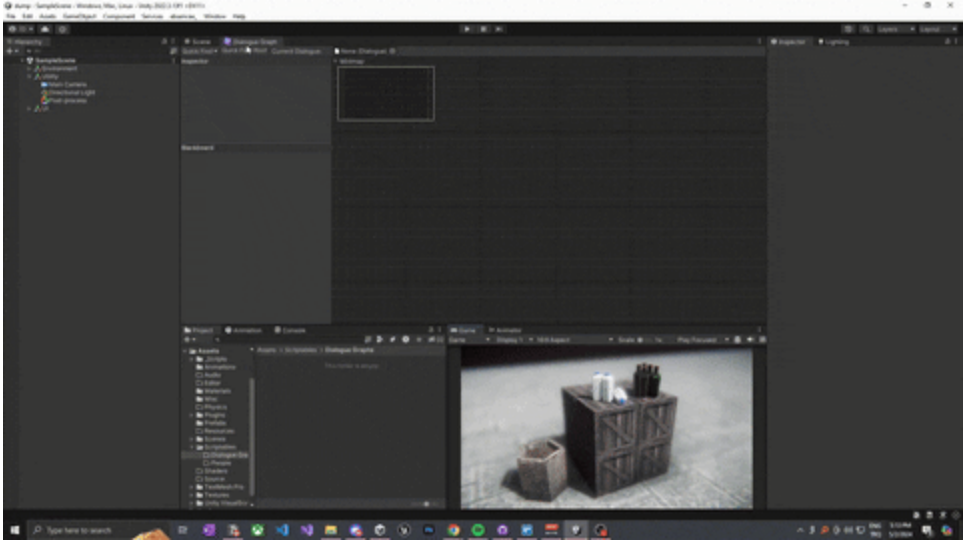
In this section of the documentation, you will see how to use this tool's editor in a pretty basic way. Let's start!

Opening up the window



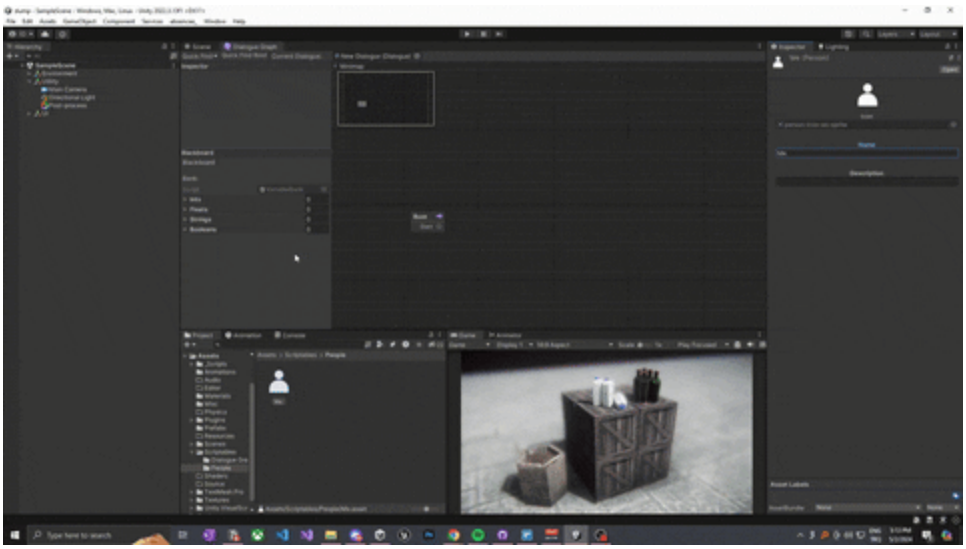
You can open up the window via the: **'absencee_/absent-dialogues/Open Dialogue Graph Window'** menu option on the Unity's toolbar.

Creating a graph



You can create a dialogue graph via the: **'absencee_/absent-dialogues/Create Dialogue Graph'** menu option on the Unity's toolbar.

Adding people to the conversation



You can add a person to a dialogue by selecting the dialogue and adding the people to the list in the inspector.

> You can toggle **graph snapping** feature via: **'Preferences->Graph Snapping'** in the unity editor.

What's Next?

Now you know the basics. You can go to Mechanism to continue reading about the tool itself. Or you can go to Demos and Examples to have a look at example use cases of the tool.

Edit this page

Demos and Examples

In this section, you will see example use cases of this tool. Let's start!

> I'll also be uploading videos on YouTube about this tool in the future.

The Built-in Demo

It takes place at the **'Plugins/absencee_/absent-dialogues/Demo'** directory, under the **Assets** folder of your current unity project.

You can open up the **'./TMP/Demo_TMP.unity'** scene and press play to give it a try.

More?

I'll be adding new demos and examples here as I develop games using this tool. You can jump to Mechanism and go on.

Edit this page

Contributing

You can contribute via **forking** or **branching** the **repository** of this package. **Please be clear with your commit descriptions.**

Or you can just point out the issues of the package via the **issues** tab of **github** instead of contributing it directly.

I appreciate any contribution from small to big, **I thank you for all you've done!**

Have a nice day!

Mechanism

In this section of the documentation, I'll be explaining you the base mechanism this tool relies on. I highly recommend you to read this section if you want to **contribute** to this tool or just wnat to **expand** the tool for your own use. And also, before starting, I recommend you to read the Basic Setup section before reading this one. Let's give it a go!

The Graph

Here is the *demo* graph that comes with the tool itself. As you can see, there are **nodes**, a **VariableBank**, and an **inspector**. You will be learning more about these in the further pages, so I won't explain all of them now.

Instead, I will be covering the **whole concept** of this tools working principle in a superficial way.

So, when you create a graph, you create a new scriptable object called a **Dialogue.** And that dialogue comes with it's own another scriptable object, and that one's called the **Blackboard VB.** Blackboard VB is the same thing with the **VariableBank**, it just has a special name for itself.

The only things you need to know about nodes for now is: they have reference the nodes connected to them by the **right-side.** So the flow is one-way only. The only node that breaks this rule is the **Goto Node.**

And as you can see, all of the nodes that you create also gets created as an usset under the dialogue you've created.

> As you may noticed, the **RootNode** got created by itself, with the dialogue. It is because the root node is a must-have node per dialogue. So, don't try to delete it.

Now you know that everything is stored as **scriptable object** assets on the disk. Let's see when you press play.

The Runtime System

Here is the demo graph again, but it is on the flow (which means it is used by a dialogue instance).

The node with the **red** outline is the current node that gets displayed on screen, while the **gray** outliend ones are the 'already seen' ones.

But as you can notice from the dialogue field at he top of this window, this is not our original **'Demo.asset'** dialogue file. This is a **clone** dialogue.

Well that's how this system works. When you're using a **Dialogue Player**, it clones the dialogue you referenced on the construction.

The purpose with this is to prevent you from losing any data on the flow. None of the changes you've made will write to the disk, they will stay on the memory.

> You can select a game object with a **Dialogue Instance** attached in order to see it's cloned dialogue live.

The Dialogue Player

Dialogue Player is the class responsible for using a dialogue. What it does is pretty simple.

1. **Clone** the **referenced dialogue** (when constructor gets called).
2. Let you progress with the **'Continue(...)'** function.
3. Let you decide what to do with the current **state**, and the **data** of the current node.

And that's it!

Dialogue Player does not perform anything by itself. It only updates it's state in order for you to know what is going on with the flow.

I highly recommend reading the API Documentation of Dialogue player for further information about the functions and the state of it.

> As said earlier, changes made during play mode (or in-game) **does not get saved by default.** So, you can write your own save logic over it, or use my absent-saves package.

What's Next?

This section is ended. Go to Nodes to continue.

Mechanism

In this section of the documentation, I'll be explaining you the base mechanism this tool relies on. I highly recommend you to read this section if you want to **contribute** to this tool or just wnat to **expand** the tool for your own use. And also, before starting, I recommend you to read the Basic Setup section before reading this one. Let's give it a go!

The Graph

Here is the *demo* graph that comes with the tool itself. As you can see, there are **nodes**, a **VariableBank**, and an **inspector**. You will be learning more about these in the further pages, so I won't explain all of them now.

Instead, I will be covering the **whole concept** of this tools working principle in a superficial way.

So, when you create a graph, you create a new scriptable object called a **Dialogue.** And that dialogue comes with it's own another scriptable object, and that one's called the **Blackboard VB.** Blackboard VB is the same thing with the **VariableBank**, it just has a special name for itself.

The only things you need to know about nodes for now is: they have reference the nodes connected to them by the **right-side.** So the flow is one-way only. The only node that breaks this rule is the **Goto Node.**

And as you can see, all of the nodes that you create also gets created as an usset under the dialogue you've created.

> As you may noticed, the **RootNode** got created by itself, with the dialogue. It is because the root node is a must-have node per dialogue. So, don't try to delete it.

Now you know that everything is stored as **scriptable object** assets on the disk. Let's see when you press play.

The Runtime System

Here is the demo graph again, but it is on the flow (which means it is used by a dialogue instance).

The node with the **red** outline is the current node that gets displayed on screen, while the **gray** outliend ones are the 'already seen' ones.

But as you can notice from the dialogue field at he top of this window, this is not our original **'Demo.asset'** dialogue file. This is a **clone** dialogue.

Well that's how this system works. When you're using a **Dialogue Player**, it clones the dialogue you referenced on the construction.

The purpose with this is to prevent you from losing any data on the flow. None of the changes you've made will write to the disk, they will stay on the memory.

> You can select a game object with a **Dialogue Instance** attached in order to see it's cloned dialogue live.

The Dialogue Player

Dialogue Player is the class responsible for using a dialogue. What it does is pretty simple.

1. **Clone** the **referenced dialogue** (when constructor gets called).
2. Let you progress with the **'Continue(...)'** function.
3. Let you decide what to do with the current **state**, and the **data** of the current node.

And that's it!

Dialogue Player does not perform anything by itself. It only updates it's state in order for you to know what is going on with the flow.

I highly recommend reading the API Documentation of Dialogue player for further information about the functions and the state of it.

> As said earlier, changes made during play mode (or in-game) **does not get saved by default.** So, you can write your own save logic over it, or use my absent-saves package.

What's Next?

This section is ended. Go to Nodes to continue.

Edit this page

Nodes

Nodes are the base elements of this tool. As I said, this is a *node based* dialogue system. So, In this section of the documentation, you will get a brief explanation of all node types included by default in this tool.

Root Node

Root Node is automatically created when you create a new graph. This node acts as a starting point in the graph. Because of that, its **UNDESTROYABLE!!!**.

Dialogue Part Node

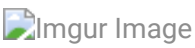
Dialogue Part Node works similar to the Root Node. This is also a starting point node **BUT** it is not the starting point of the graph itself. It is a starting point of the node chain it is connected to.

Goto Node

Goto Node is used to seperate the node chains to have a more clear graph window. The only mission of this node is to find the target **Dialogue Part Node** and teleport to it.

> The relation between the **Goto Node** and **Dialogue Part Node** is *string prone* for now. So, be careful while using them.

Fast Speech Node



Fast Speech Node is one of the two speech nodes in this tool. This node is used to display a speech that has no options.

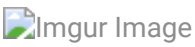
Decision Speech Node

Decision Speech Node does the same thing with the Fast Speech Node. The only difference between them is that the Decision Speech Node has options the player acn choose from.

Option Block

Option Block is not a Node itself. It is used to display the options of a Decision Speech Node. Text written in the text field of this block will be displayed as an option.

Condition Node



Condition Node lets you to progress in different ways in the dialogue depending on some conditions. The conditions are based on the **Variable Comparers** like the ones in the Option Block.

Action Node

Action Node is pretty self explanatory. It invokes some actions when it gets *reached* by the dialogue.

> There is also a property of Action Node called CustomAction() which is a virtual method. And also **Action Node** one of the two node subtypes which you can derive new subtypes from (they aren't sealed). So, you can derive from Action Node to have a node that has some more specific actions to perform.

Sticky Note Node

StickyNote is not really a node. It is derived from the node base type and that's all. It has no effect on the dialogue flow. But you can use it to leave some useful notes in the graph view.

Title Node

Title Node is nearly the same as Sticky Note Node. It is just bigger and easy to see.

What's Next?

This section is ended. Go to Components to continue.

Edit this page

Components

Components are essential when you're working with a tool which is integrated right into unity. In this section of the documentation, you will get a simple explanation of built-in components that come this tool.

Dialogue Displayer

Dialogue Displayer is the only built-in way of displaying a dialogue on screen. But you can come up with your own solutions.

> **Dialogue Displayer** is a singleton! Use it with this knowledge.

Option Text

Option Text is a needed component when working with the built-int Dialogue Displayer It is simply responsible for the *clicking* action, *index* holding and of course, displaying the option.

> **Dialogue Instance** is designed to work with the Dialogue Player. You can read Mechanism page for more details about that class.

Dialogue Instance

Dialogue Instance is the best way to play a dialogue in your game. It is pretty straight-forward to use. All you have to do is: attach, drag, press play. And you're done!

> The *player* section below in the inspector is only visible in **play mode.**

Dialogue Sounds Player

Dialogue Sounds Player is the *extension* component responsible for playing sounds from an audio source, using the information from AdditionalSpeechData.

Dialogue Animations Player

Dialogue Animations Player is the *extension* component responsible for managing animator controller over a dialogue, using the information from AdditionalSpeechData.

Dialogue Input Handler (Legacy)

Dialogue Input Handler (Legacy) is the *extension* component responsible for handling the input comes from player during the dialogue. It is marked **legacy** because it only works with the old input system of unity.

> All of the *extension* components above are derived from the DialogueExtensionBase class. See Custom Dialogue Extensions for more information..

What's Next?

Well, you know everything you need to know about this tool right now. But if you want to modify it for your own use, you can go to Advanced to see how you can do it easily.

Custom Nodes

In this section of the documentation, you will learn how to create your own custom nodes. Let's start.

When creating custom nodes, you have three options:

1. Using the **Node** class as base.
2. Using the **ConditionNode** class as base.
3. Using the **ActionNode** class as base.

Well, the **2nd** and the **3rd** options will come with their limitations. Only a little part of their inheritable members are editable. So, if you want to create a unique node, best option is the **1st** one.

You can find a more detailed information about **2nd** and **3rd** options in the API Documentation.

Using Node as Base

Well, the node class has a lot to inherit from. You can find a better list in it's API Documentation section. You can use those metyhodsand properties to create any node you want.

> If you want to create a node which has some really unique features, you might need to modify the NodeView class. This is only necessary if your custom node needs to display or hide some data in the graph.

What's Next?

This section is ended. Go to Custom Dialogue Extensions to continue.

Custom Nodes

In this section of the documentation, you will learn how to create your own custom nodes. Let's start.

When creating custom nodes, you have three options:

1. Using the **Node** class as base.
2. Using the **ConditionNode** class as base.
3. Using the **ActionNode** class as base.

Well, the **2nd** and the **3rd** options will come with their limitations. Only a little part of their inheritable members are editable. So, if you want to create a unique node, best option is the **1st** one.

You can find a more detailed information about **2nd** and **3rd** options in the API Documentation.

Using Node as Base

Well, the node class has a lot to inherit from. You can find a better list in it's API Documentation section. You can use those metyhodsand properties to create any node you want.

> If you want to create a node which has some really unique features, you might need to modify the NodeView class. This is only necessary if your custom node needs to display or hide some data in the graph.

What's Next?

This section is ended. Go to Custom Dialogue Extensions to continue.

Edit this page

Custom Dialogue Extensions

In this section of the documentation, you will learn how to write your own dialogue extensions. Let's start.

To create custom dialogue extensions, you have to inherit the **DialogueExtensionBase** class.

```
#if UNITY_EDITOR
        [UnityEditor.MenuItem("CONTEXT/DialogueInstance/Add Extension/EXTENSION_NAME")]
        static void AddExtensionMenuItem(UnityEditor.MenuCommand command)
        {
            DialogueInstance instance = (DialogueInstance)command.context;
            instance.AddExtension<EXTENSION_TYPE>();
        }
#endif
```

What's Next?

You've done it! You've read all the way to the end, mate. I really appreciate you. Know you don't just know how this tool works in a detailed way but you also know how you can modify it.

There is nothing else you need to read. But if you want to know how you can publish any modifications you've made on the tool to the open-source community, I'd suggest you reading Contributing section.

And again, thank you for reading this all the way!

Have a nice day!

Edit this page

Roadmap

In this section of the documentation, you will see some details about the past and the future of this project.

Latest Version (v0.3.1-alpha)

- Small bugs fixed.
- Editor window consistency is far better from now on.
- Restricted the usages of VariableBanks with nodes for a healthier structure.

Upcoming Version (v1.0.0)

- Dialogue editor refresh automation.
- Stabler graph view.
- Grid snapping fix (will be added to the docs).
- Bugfix of DialogueInputHandler (Legacy) checking for input event if the instance is not in the dialogue.
- Rewrite of the Option system (now it supports multiple comparers for visibility).
- Switching to the latest version of absent-variables **(not backwards capable).**
- Handling the runtime displaying of the graph.
- Refined code.
- Folder restrucure.
- Custom documentation for components and nodes.
- More and better API documentation.

Already on the Plans

- Rewriting the GotoNode and geting rid of string prone code.
- Localization for dialogues.

- Better documentation for API and the editor members.
- New images and gifs for the documentation.
- New icons for the components, nodes and etc.
- An extension based on the absent-saves package.

In the Future (Without Certainty)

- Import/Export/Backup support.
- Integration with some other applications.
- Generic option feature.
- Dialogue override graph feature.

Namespace com.absence.dialoguesystem

Classes

Dialogue

The scriptable object derived type that holds all of the data which is essential for a dialogue.

DialogueAnimationsPlayer

A small component which is responsible for playing the animations (if there is any) of the dialogue instance attached to the same game object.

DialogueDisplayer

A singleton with the duty of displaying the current dialogue context. Written for the Unity UI package. Not compatible with the UI Toolkit.

DialogueExtensionBase

This is the base class to derive from in order to handle some custom logic over the system.

DialogueInputHandler_Legacy

A small component with the responsibility of using the input comes from player (uses legacy input system of unity) on the dialogue.

DialogueInstance

Lets you manage a single `DialoguePlayer` in the scene easily.

DialogueOptionText

A small component that manages the functionality of an option's drawing and input.

DialoguePlayer

Lets you progress in a dialogue easily.

DialogueSoundsPlayer

A small component which is responsible for playing the sounds (if there is any) of the `DialogueInstance` attached to the same gameobject.

Interfaces

IUseDialogueInScene

Any game object with a script that implements this interface attached will display it's dialogue when gets selected.

Enums

DialoguePlayer.PlayerState

Shows what state the dialogue player is in.

Delegates

DialogueInstance.SpeechEventHandler

The delegate responsible for handling events directly about speech.