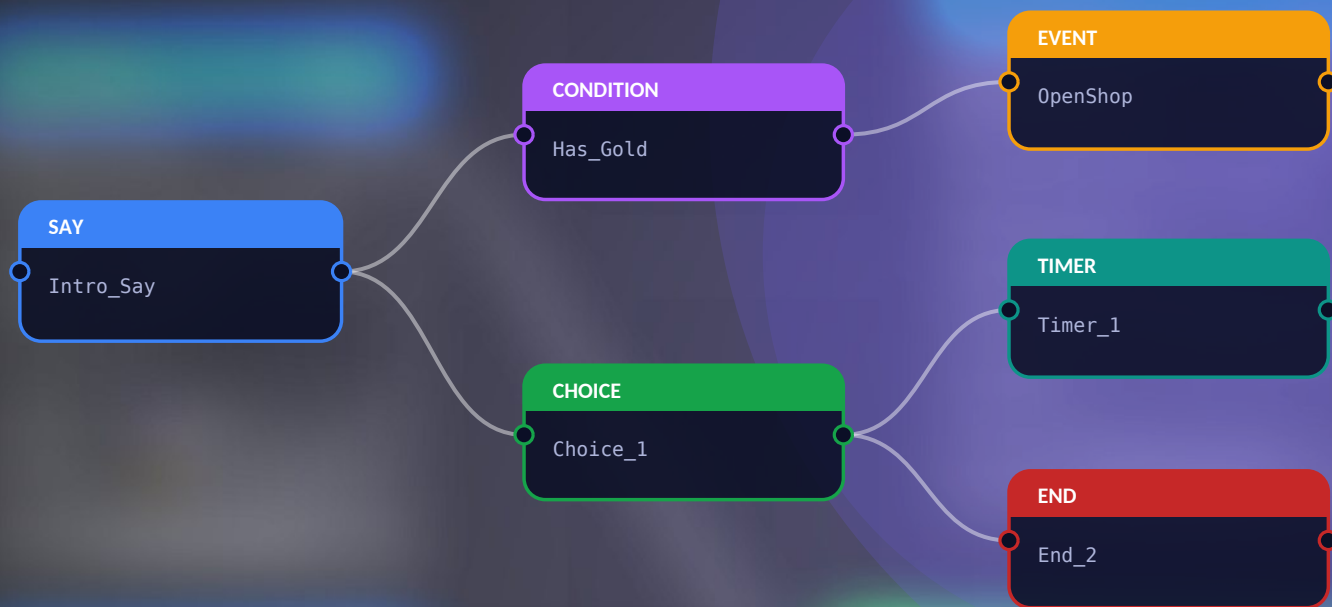




Dialogue System

Easy & Pro

Complete reference — graph editor, runtime,
localization and Excel workflow

Graph Editor

Runtime

Localization

Excel Workflow

Contents

A node-based dialogue system for Unity: visual graph editor, runtime player, and built-in multi-language localization with Excel import/export.

01	Quick Start Six steps to your first dialogue	3	07	Localization Translations stored on the nodes	10
02	Core Concepts Graphs, node names, ports	4	08	Excel Workflow Export, translate, import	11
03	Node Types Say, Choice, Event, Condition, End, Timer, Random	5	09	UI Setup Wizard One-click dialogue prefab	12
04	Variables & Conditions State, flags and branching rules	6	10	Code Examples Copy-paste snippets	12
05	Runtime DialogueUI, events, runner, trigger	7	11	Tips & Gotchas Things that bite	17
06	Graph Editor Canvas, toolbar, shortcuts	9			

Unity 2021 LTS+

Editor uses GraphView and UI Toolkit, both stable since 2021.

TextMeshPro

Import TMP Essentials when prompted (Window → TextMeshPro). Needed for TMP_Text.

No extra DLLs

Excel I/O is built on System.IO.Compression + System.Xml.Linq. Nothing to install.

01 Quick Start

Six steps to your first dialogue

1

Create a graph

Window → Dialogue System → Graph Editor, then Create. A new graph opens with one Say node.

2

Build the flow

Right-click or Ctrl+Space to add nodes. Drag from an output port to an input port.

3

Write the text

Click a Say node and type the Speaker and Text straight into its body.

4

Save

Click Save in the toolbar. There is no autosave and no Ctrl+S binding.

5

Build a UI

Window → Dialogue System → Create Dialogue UI generates a fully wired prefab.

6

Play it

Add a DialogueTrigger, set UI + graph, call `Interact()` or tick `triggerOnStart`.

■ What's in the box

File	Role
DialogueGraph.cs	Graph asset (ScriptableObject) and node connections
DialogueNode.cs	All node types: Say, Choice, Event, Condition, End, Timer, Random
DialogueChoice.cs	Choice options, conditions, session variables
DialogueLocalization.cs	Runtime locale tracking and TranslationSet
DialogueRunner.cs	Optional shared settings / variables / event hub
DialogueUI.cs	Runtime player that drives a graph on screen
DialogueUIAutoSetup.cs	Editor wizard that builds a ready-made UI prefab
Editor/	Graph window, canvas, node views, search, Excel utilities, USS stylesheet. Editor-only; keep it inside a folder literally named <code>Editor</code> .

02 Core Concepts

Graphs, node names, ports

A `DialogueGraph` is a `ScriptableObject` (Create → Dialogue System → Dialogue Graph) holding a polymorphic `nodes` list, a flat `connections` list (from/to GUID + port index), a `startNodeGuid`, and metadata (`dialogueName`, `description`, `category`).

KEY IDEA Every node has a hidden GUID and a readable, unique **Node Name For Coding** (`Say`, `Say_2`, `Choice_3`). Rename it freely in the Inspector, and always use this name, never the GUID, from code and Excel.

NOTE Each node has one input port (In) and one or more outputs. An output feeds exactly one destination: connecting it again replaces the old wire.

Method	What it does
<code>GetNode(guid)</code>	Look up a node by GUID
<code>Node(name) / TryGetNode(...)</code>	Look up a node by its coding name
<code>StartNode</code>	The configured start node (or the first node if none is set)
<code>GetConnectedNodes(guid, port)</code>	All nodes wired to an output port
<code>GetNextNode(guid, port)</code>	The single next node for a port. This is what the runtime uses
<code>Validate()</code>	List of problems: empty graph, no start node, Choice without choices, duplicate/empty names

03 Node Types

Seven building blocks, color-coded in the editor

All nodes derive from `DialogueNode` (`guid`, `position`, `nodeNameForCoding`, `NodeType`, and an `OnEnter(runner)` extension point that the built-in runtime does not use).

Say

1 input
1 output: Out

NPC line. Sets nameplate, portrait and audio, types the text, then waits for Space / Left-Click.
`text`, `speakerName`: source text (+ per-language translations)
`portrait` `Sprite` · `voiceLine` `AudioClip`
`animationTrigger`: not auto-fired; read it in `onNodeStarted`

Choice

1 input
One output per choice (index = i)

Player choice. Shows a prompt, then buttons; flow continues down the port of the picked button.
`promptText`, `speakerName`, `choices[]`
`autoSelectTime`: >0 auto-picks port 0 after N s; 0 waits forever
`buttonLayout` Left/Right/Top/Bottom (default Top) · `buttonSpacing` (default 2)
Each choice: `text`, `visibilityCondition`, `showIfNotMet` (greyed out instead of hidden)

Event

1 input
1 output: Out

Pass-through call into your game code: camera shake, quest update, cutscene, anything.
`eventName`: the name your handler registers for · `eventArg`: free-form string
`waitForCompletion`: true pauses playback until the handler calls `done()`

Condition

1 input
2 outputs:
0 True · 1 False

Branch logic. Evaluates a list of conditions and takes the True or False port.
`conditions[]` · `mode`: All (AND) or Any (OR)

End

1 input
no outputs

Terminates the graph, fires `onDialogueEnded` and hides the panel. A graph may have several Ends.
No fields

Timer

1 input
2 outputs:
0 Done · 1 Skip

Wait, then branch. Optionally skippable by the player.
`duration` · `showCountdown` · `countdownText` (use `{time}` for remaining seconds)
`skippable` · `skipButtonText`: if empty, Space/Click skips instead of a button

Random

1 input
portCount outputs (labeled with %)

Random branch. Picks one output port each time it is reached.
`portCount` (min 2) · `weighted` · `weights[]` (higher = more likely)

04 Variables & Conditions

State, flags and branching rules

DialogueVariables is a lightweight key → string store, one per DialogueRunner (or per DialogueUI without a runner).

Method	Purpose
Set(key, value)	Session-scoped variable
SetGlobal(key, value)	Static, process-wide. Shared by every instance. Use for persistent flags like quest state
TryGet(key, out value)	Session value first, then global fallback
Increment(key, delta = 1)	Parses the value as float, adds delta, stores it back
Format(text)	Replaces every {name} with its value (automatic in Say / Choice text)
SaveGlobals() · LoadGlobals(dict) · ClearGlobals()	Hooks for your own save system

TIP A Say node with You have {gold} gold. substitutes the live value automatically, in whichever language is active.

■ DialogueCondition

Fields: variableName, op (Equals, NotEquals, GreaterThan, LessThan, GreaterOrEqual, LessOrEqual, Contains) and value.

- Comparison is numeric when both sides parse as floats, otherwise string. Contains is always a substring check.
- A variable that doesn't exist evaluates to false.
- DialogueCondition.EvaluateAll(list, mode, vars) powers Condition nodes, DialogueTrigger.startConditions and choice visibility.

05 Runtime

DialogueUI, events, runner, trigger

■ DialogueUI

The main runtime controller. Attach one to your dialogue Canvas (or let the wizard do it). Each UI is fully self-contained, so several can coexist, e.g. one per split-screen player.

Inspector field	Purpose
dialoguePanel	GameObject shown/hidden while dialogue runs
speakerNameText, dialogueBodyText	TMP_Text references
portraitImage, nameplateBackground	Portrait sprite; nameplate hidden when no speaker is set
choicesContainer, choiceButtonPrefab	Parent for runtime-spawned buttons; prefab = Button + child TMP_Text
charsPerSecond, typingSound	Typewriter speed and optional per-character sound
pauseGameDuringDialogue	Time.timeScale = 0 while running (a shared runner's setting wins)
skipToEndOnInput	Space/Click mid-typing completes the line instead of being ignored

API	Purpose
StartDialogue(graph) · StopDialogue()	Begin from the start node · force-stop (fires onDialogueEnded)
IsRunning · CurrentNode · Variables	Current state; Variables is the runner's or the UI's own
RegisterEventHandler(name, h) / Unregister...	Hook up Event nodes
Node(nodeName)	Returns { onStarted, onEnded } UnityEvents for that one node

■ Events and firing order

Event	Fires when	Typical use
onDialogueStarted	StartDialogue() called, before the graph begins	Freeze player, open dialogue camera
onDialogueEnded	End node reached or StopDialogue()	Resume control, save progress
onNodeStarted(name)	Every node, on entry	Logging, analytics, drive an Animator
onNodeEnded(name)	Every node, after its behavior, before the next	Per-node cleanup
Node("X").onStarted / .onEnded	Only for node X	Give this quest, play this sound



NOTE Condition and Random nodes have no waiting behavior, so their started/ended events fire back-to-back on the same frame. `onDialogueEnded` always fires after the last `onNodeEnded`. In the Editor, `Validate()` runs on every `StartDialogue()` and logs warnings (non-blocking).

TIP Use the **global** events for "every node the same way" (logging, a UI flash). Use `Node("Name")` for one special node. It scales far better than an `if (nodeName == "...")` chain.

■ DialogueRunner, DialogueTrigger, TypewriterEffect

- `DialogueRunner` is an optional shared hub. Link it via `DialogueUI.sharedRunner` so several UIs share Variables and Event handlers. Its `pauseGameDuringDialogue` / `skipToEndOnInput` override the UI's, but a UI's own handler for an event name wins over the runner's.
- `TypewriterEffect` is auto-added to the `DialogueUI` `GameObject`; you rarely touch it.

DialogueTrigger field	Purpose
<code>dialogueUI</code>	Target UI (auto-found via <code>FindAnyObjectByType</code> in <code>Awake</code> if empty)
<code>graph</code>	The <code>DialogueGraph</code> to play
<code>triggerOnStart</code>	Fire automatically in <code>Start()</code>
<code>triggerOnTrigger.interactionTag</code>	Fire on <code>OnTriggerEnter</code> (needs a trigger Collider); tag default "Player"
<code>startConditions.conditionMode</code>	Gate the trigger behind conditions

Call `Interact()` or `TryStartDialogue()` yourself for scripted starts. The trigger refuses to start while `IsRunning` is true.

■ Choice button layout

`ApplyChoiceLayout` disables any `LayoutGroup` on `choicesContainer`, measures each button, then stacks them from the container pivot in the chosen direction (Left / Right / Top / Bottom) with `buttonSpacing` plus the largest button size, so buttons never overlap. Default is Top / 2 px.

06 Graph Editor

Canvas, toolbar, shortcuts

Open via Window → Dialogue System → Graph Editor, or double-click any `DialogueGraph` asset.

- **Left panel:** every graph in the project, grouped in collapsible folders. Create / Load / Refresh buttons, a x per row to delete (with confirmation), and a green badge for the category.
- **Right panel:** the canvas, with a MiniMap pinned top-left.

Toolbar	Action
Save	Writes node positions and connections into the asset, then <code>AssetDatabase.SaveAssets()</code>
Validate	Runs <code>DialogueGraph.Validate()</code> and shows the result
Language (globe)	Language picker, plus Excel export/import (see §07-08)
Auto Layout	Tidy columns via breadth-first walk from the start node; unreachable nodes stay put
Center	Frames the whole graph (<code>FrameAll</code>)
Undo / Redo · Delete	Undo/redo; delete the open graph asset (confirmed, permanent)

■ Creating and wiring nodes

- Right-click the canvas → Add Node, or press **Ctrl+Space** (or drop a dragged wire on empty space) for the searchable node list: Basic (Say, Choice, End) · Logic (Condition, Event) · Flow (Timer, Random).
- Drag from an output dot to an input dot. Compatible ports highlight; a port can't connect to itself or to the same side.
- Deleting a node removes its connections. Deleting the start node clears `startNodeGuid` (falls back to the first node).
- Deleting a choice removes its port and shifts later wires down, so there are never gaps in port indices.
- **Copy/paste** (Ctrl+C / Ctrl+V) gives new GUIDs and unique names; wires between copied nodes are kept, wires to outside nodes are not.
- Every structural edit is wrapped in `Undo.RecordObject`: Ctrl+Z, Ctrl+Y / Ctrl+Shift+Z.

■ Node view

Color-coded

Blue Say, green Choice, orange Event, purple Condition, dark red End, teal Timer, violet Random. Start node gets a highlighted border.

Collapsible + preview

The Arrow button in the title bar collapses a node. A live preview line shows roughly what it displays at runtime.

Language badge

Editing a non-English language shows a colored badge (e.g. FR) on every translatable node.

WATCH OUT **Save is manual.** Canvas edits live only in the open `GraphView` until you click Save. It's easy to forget before closing the window.

Selecting a graph asset in the Project window shows an Inspector with **Open in Graph Editor**, **Validate Graph**, and the same Excel export/import buttons.

07 Localization

Translations stored right on the nodes

KEY IDEA No key tables to manage. The text you type into a node is the English source. Every other language is an optional override stored on the same node in a `TranslationSet`. Nothing to sync, nothing external to lose.

Node	English field	Translation field
SayNode	text	textTranslations
SayNode	speakerName	speakerTranslations
ChoiceNode	promptText	promptTranslations
ChoiceNode	speakerName	speakerTranslations
DialogueChoice	text	textTranslations

At runtime, `GetFormattedText`, `GetLocalizedSpeaker`, `GetLocalizedPrompt` and `DialogueChoice.GetText` resolve against `LocalizationSystem.CurrentLocale`, fall back to English, then apply `{variable}` substitution. `DialogueUI` needs no changes.

```

LocalizationSystem
LocalizationSystem.CurrentLocale // "en", "fr", ... (default "en")
LocalizationSystem.SetLocale("fr"); // switch instantly, nothing reloads
LocalizationSystem.OnLocaleChanged += RefreshOpenDialogueIfAny;
LocalizationSystem.DefaultLocale // constant "en"

```

- `DialogueLocalizationLoader` (optional) picks the starting language: `startLocale` forces one, empty auto-detects via `SystemLanguageToLocale`; `rememberChoice` persists to `PlayerPrefs`. Call `loader.SetLocale("de")` from your settings menu.

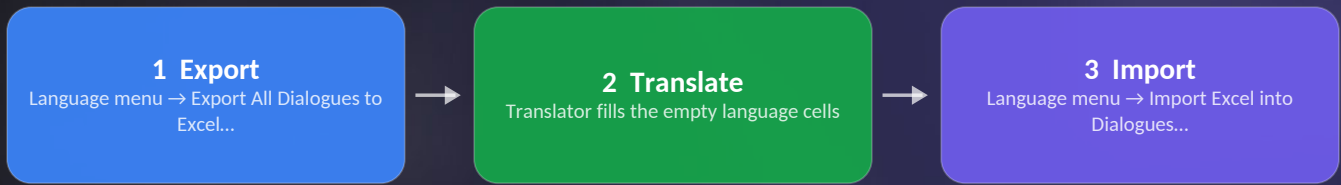
■ Language menu (globe icon)

Entry	What it does
Edit in / <language>	Repopulates the canvas so every text field shows and edits that language
Add Language...	~28 presets (French, German, Spanish, Japanese, Arabic...) or a custom code; switches you into it
Remove Language / <language>	Hides it from the picker. Stored translations are kept until you re-add the code
Export / Import Excel	See next section

- English (en) is always present and can't be removed. The project language list lives in an auto-created asset at `Assets/Editor/EasyProDialogues/DialogueLocalizationSettings`.
- In a non-English language, labels read **Text [FR]**, fields get a light-blue border, and an empty field shows a gray `EN: ...` hint. Clearing a field removes that override (runtime falls back to English).

08 Excel Workflow

Export, translate, import



Export scans every graph in the project into a single **Translations** sheet. Translators only fill empty cells in the language columns.

Graph	Node	Field	Index	en (default)	fr	de
Chapter1	Intro_Say	Text		Hello, traveler!		
Chapter1	Intro_Say	Speaker		Old Man		
Chapter1	Choice_1	Prompt		What do you want?		
Chapter1	Choice_1	Choice	0	Tell me about the ruins		
Chapter1	Choice_1	Choice	1	Nothing, goodbye		

- **Graph** is the asset file name (guaranteed unique); **Node** is the coding name; **Field** is Text, Speaker, Prompt or Choice; **Index** is the 0-based choice number.

■ Import rules

- Rows match by Graph + Node + Field (+ Index). Renamed or deleted nodes are skipped, and the dialog reports the count.
- A non-empty `en (default)` cell overwrites the English source.
- Each language column overwrites that translation, **including empty cells**, which clear it.
- Unknown headers are treated as new language codes and auto-added (e.g. a translator adds `pt-BR`).
- Column order doesn't matter (matched by header). Extra helper columns like `Notes` are ignored.

WATCH OUT Keep **Node Name For Coding** stable once files are out for translation. A rename breaks the row match (the translation itself isn't lost, but you'll need to re-export).

NOTE The output is a real, standards-compliant `.xlsx` (Excel, Google Sheets, LibreOffice, Numbers). Repeated round-trips are safe. You can also translate directly in the editor and skip Excel entirely.

09 UI Setup Wizard

One-click dialogue prefab

Window → **Dialogue System** → **Create Dialogue UI** builds a complete, styled dialogue Canvas in the active scene, saves it as a prefab (plus a **ChoiceButton** sub-prefab) and wires every **DialogueUI** field for you.

- **Options:** save path, panel height (% of screen), include portrait frame, Animator Controller slot.
- **Colors:** panel background, nameplate, body text, speaker text, choice button (normal), choice button text.

Generated hierarchy

```
DialogueCanvas (Canvas, CanvasScaler, GraphicRaycaster, AudioSource, DialogueUI)
└─ DialoguePanel
  └─ Nameplate → SpeakerNameText
  └─ PortraitFrame → PortraitImage (if enabled)
  └─ DialogueBodyText
  └─ ContinueIndicator (▼)
  └─ ChoicesContainer
    └─ (ChoiceButton instances created at runtime)
```

10 Code Examples

Copy-paste snippets

Copy-paste snippets. Assume using `EasyProDialogues`;

■ Start a dialogue from script

NpcInteractable.cs

```
using EasyProDialogues;
using UnityEngine;

public class NpcInteractable : MonoBehaviour
{
    [SerializeField] DialogueUI dialogueUI;
    [SerializeField] DialogueGraph graph;

    public void OnPlayerInteract()
    {
        if (dialogueUI.IsRunning) return; // already talking
        dialogueUI.StartDialogue(graph);
    }
}
```

■ Dialogue, global and named node events

DialogueEventsExample.cs

```
void OnEnable()
{
    // Dialogue-level: once per StartDialogue() / EndNode
    dialogueUI.onDialogueStarted.AddListener(() => Debug.Log("Started"));
    dialogueUI.onDialogueEnded.AddListener(() => Debug.Log("Ended"));

    // Global: fires for EVERY node, passing its coding name
    dialogueUI.onNodeStarted.AddListener(n => Debug.Log($"Entered {n}"));
    dialogueUI.onNodeEnded.AddListener(n => Debug.Log($"Finished {n}"));

    // Named: fires for ONE node only
    var quest = dialogueUI.Node("Quest_Accepted");
    quest.onStarted.AddListener(() => QuestLog.Add("main_quest"));
    quest.onEnded.AddListener(() => AudioManager.PlayStinger("quest_started"));
}
```

Use named methods instead of lambdas if you need `RemoveListener` in `OnDisable`.

■ Event node handlers

ShopkeeperEvents.cs

```
void Awake()
{
    // Fire-and-forget (waitForCompletion = false): `done` is null, don't call it
    dialogueUI.RegisterEventHandler("ShakeCamera", (evNode, done) =>
    {
        cameraShake.Shake(float.Parse(evNode.eventArg));
    });

    // Wait-for-completion (waitForCompletion = true): playback pauses until done()
    dialogueUI.RegisterEventHandler("OpenShop", (evNode, done) =>
    {
        shopUI.Open(evNode.eventArg, onClosed: () => done?.Invoke());
    });
}

void OnDestroy()
{
    dialogueUI.UnregisterEventHandler("ShakeCamera");
    dialogueUI.UnregisterEventHandler("OpenShop");
}
```

■ Variables and saving globals

DialogueSaveSystem.cs

```
dialogueUI.Variables.Set("gold", "150"); // session
dialogueUI.Variables.Increment("gold", 50);
dialogueUI.Variables.SetGlobal("quest_blacksmith_done", "true"); // shared by every instance

// JsonUtility can't serialize Dictionary, so wrap it in two parallel lists
[System.Serializable] class SaveData { public List<string> keys = new(), values = new(); }

public static void Save()
{
    var s = new SaveData();
    foreach (var kv in DialogueVariables.SaveGlobals()) { s.keys.Add(kv.Key); s.values.Add(kv.Value); }
    PlayerPrefs.SetString("dialogue_globals", JsonUtility.ToJson(s));
}

public static void Load()
{
    if (!PlayerPrefs.HasKey("dialogue_globals")) return;
    var s = JsonUtility.FromJson<SaveData>(PlayerPrefs.GetString("dialogue_globals"));
    var d = new Dictionary<string, string>();
    for (int i = 0; i < s.keys.Count; i++) d[s.keys[i]] = s.values[i];
    DialogueVariables.LoadGlobals(d);
}
```

■ Language switcher

LanguageDropdown.cs

```
[SerializeField] Dropdown dropdown; // English, French, German
[SerializeField] DialogueLocalizationLoader loader;
readonly string[] codes = { "en", "fr", "de" }; // same order as the options

void Start()
{
    int i = System.Array.IndexOf(codes, LocalizationSystem.CurrentLocale);
    if (i >= 0) dropdown.SetValueWithoutNotify(i);
    dropdown.onValueChanged.AddListener(idx => loader.SetLocale(codes[idx])); // switch + persist
}
```

■ Shared runner across several UIs

Useful for split-screen co-op, or separate "NPC talk" and "narrator caption" UIs that share quest flags and handlers.

GameBootstrap.cs

```

void Awake()
{
    player1UI.sharedRunner = sharedRunner;
    player2UI.sharedRunner = sharedRunner;

    // Registered once, available to whichever UI reaches the EventNode
    sharedRunner.RegisterEventHandler("GlobalAnnounce", (evNode, done) =>
        Debug.Log($"Announcement: {evNode.eventArg}"));
}

```

■ Full example: a shopkeeper mini-scene



Shopkeeper.cs

```

void Awake()
{
    dialogueUI.RegisterEventHandler("BuyPotion", (evNode, done) =>
    {
        dialogueUI.Variables.Increment("gold", -10);
        inventory.Add("potion", 1); // waitForCompletion = false, so no done()
    });
}

public void Talk()
{
    dialogueUI.Variables.Set("gold", inventory.Gold.ToString());
    dialogueUI.StartDialogue(shopGraph);
}

```

■ Editor scripts

● ● ● BulkUpdateChoiceLayout.cs (Editor)

```
#if UNITY_EDITOR
using UnityEditor;
using EasyProDialogues;

public static class BulkUpdateChoiceLayout
{
    [MenuItem("Tools/Dialogue System/Set All Choices To Top+2px")]
    public static void Run()
    {
        foreach (var graph in DialogueGraph.GetAllGraphs())
        {
            foreach (var node in graph.nodes)
            {
                if (node is not ChoiceNode c) continue;
                c.buttonLayout = ButtonLayoutPosition.Top;
                c.buttonSpacing = 2f;
            }
            EditorUtility.SetDirty(graph);
        }
        AssetDatabase.SaveAssets();
    }
}
#endif
```

● ● ● Excel from a build pipeline / CI (Editor)

```
#if UNITY_EDITOR
const string path = "Translations/AllText.xlsx";
int rows = DialogueLocalizationEditorUtility.ExportAllGraphsToExcel(path);
var (matched, skipped) = DialogueLocalizationEditorUtility.ImportExcelIntoGraphs(path);
Debug.Log($"Imported {matched}, skipped {skipped}");
#endif
```

■ Build a DialogueTrigger from code

● ● ● NpcSpawner.cs

```
var npc = new GameObject("NPC");
var col = npc.AddComponent<SphereCollider>(); col.isTrigger = true; col.radius = 2f;
var trigger = npc.AddComponent<DialogueTrigger>();
trigger.dialogueUI = dialogueUI; trigger.graph = greetingGraph;
trigger.triggerOnTrigger = true; trigger.interactionTag = "Player";
```

11 Tips & Gotchas

Things that bite

WATCH OUT New defaults only affect new nodes. Changing `ChoiceNode.buttonLayout` / `buttonSpacing` in code doesn't touch existing data. Edit by hand, or run the bulk-update Editor script from §10.

WATCH OUT Renaming a coding name is safe for the graph (wires use GUIDs) but breaks `runner.Node("OldName")` calls and any not-yet-re-exported Excel rows.

WATCH OUT A `ChoiceNode` with zero choices fails `Validate()`, produces no buttons, and the coroutine hangs waiting for a pick. Always give it at least one choice.

WATCH OUT Save is not automatic in the graph editor. Click Save before closing the window.

WATCH OUT Deleting a graph asset is permanent. It bypasses Unity's Trash. There's a confirmation, but no undo afterwards.

NOTE `LocalizationSystem.SetLocale` does nothing for the already-active locale, so `OnLocaleChanged` fires only on a real change.

NOTE `RandomNode` weights: with `weighted = true` and a short list, missing entries default to 1. All-zero weights fall back to uniform.

NOTE Timer skip listens for Space/Left-Click only when `skipButtonText` is empty. Set the text for an explicit on-screen skip button.