Tutorial Manager

The onboarding process is the most critical part of your game. Players who churn won't be around to experience all your great content, let alone monetize. The traditional approach – for those who even get this far – has been to optimize the tutorial for the majority of players, yielding at best a one-size-fits-all onboarding flow. At the same time, tutorials themselves tend to be developed at the last minute in a project, becoming a brittle pile of spaghetti code.

We can do better:

- Leveraging our vast pool of game data, Unity can differentiate a player who needs a little more help from the player who wants to charge straight into the action. Unity Analytics collects data as players play your game and trains a machine learning model to predict whether individuals like your game better with more help or less help.

- To help you avoid writing last minute, spaghetti code, the Tutorial Manager provides a state-machine-based API. This state machine lets you define the high-level structure of your tutorials and then advance the state as the player completes a tutorial using a few, simple API calls.

- The Tutorial Manager dashboard helps you manage your live tutorials, update tutorial text remotely, and view reports on tutorial effectiveness.

With just a quick integration, the Tutorial Manager can put you on the road to optimizing your game one player at time!

Release Notes

The current release is v0.1.0. The following are known issues with this release:

- Some users have reported that an excessive number of tutorials and steps (far more than we anticipate being useful in real-world cases) can result in data not being uploaded at author-time.
- Some users have reported an intermittent token authorization issue. This problem manifests itself when pushing or pulling data, and can result in a failure of the operation. Look for 401 and 404 errors in the output console. To workaround, first try closing and opening the Tutorial Editor. If this fails, it may be necessary to restart Unity.

Contents

Getting Started

Tutorial Manager is supported for PC, iOS and Android projects. You can use Unity 5.5 or newer. You must enable Unity Analytics.

1. Import the Tutorial Manager Asset Package into your project (menu: **Assets** > **Import Package** > **Custom Package**).
2. If not already enabled, turn on Unity Analytics from the Services window (menu: **Window** > **Services**).
3. Create your tutorial GameObjects… For tips on designing tutorials that work with the Tutorial Manager, see Designing Tutorials for Adaptability.
4. Open the **Tutorial Editor** window (menu: **Window** > **Unity Analytics** > **Tutorial Editor**).
5. Enter the Tutorial structure into the window. See Creating the Tutorial State Model.
6. Add AdaptiveContent components to any GameObjects that should be turned on when you play a tutorial step and off when the step completes. See Add Adaptive Components to Your Tutorials.
7. Add AdaptiveText components to any tutorial GameObjects with Text, Text Mesh or Text Mesh Pro components. (Note that you should only use one AdaptiveText component per tutorial step.)
8. Click the **Push Data** button on the **Tutorial Editor** window to send your tutorial data to the Analytics service.
9. Add code to play your tutorials at appropriate points in your game. See Playing Tutorials.

Designing Tutorials for Adaptability

Consider the following when planning your tutorial code:

- Adaptive components work by turning game objects on (`gameObject.SetActive(true)`) when a tutorial step starts and turning them off when that step is complete.
- You can initiate any dynamic sequences within a step using MonoBehaviour events. For example, Unity invokes OnEnable when a GameObject is turned on.
- You can bind adaptive components to more than one tutorial step to share GameObjects across multiple steps and multiple tutorials.
- Child GameObjects are only enabled when their parent GameObject is enabled, so be careful when nesting adaptive content.
- The Tutorial Manager automatically creates a remote text field for AdaptiveText components, which you can use to revise the text even after your game is live.
- Only one AdaptiveText component is supported per tutorial step. You can use AdaptiveContent components for additional tutorial text elements, but remote fields are not created for such elements.
- AdaptiveContent components have an option, **Respect 'off' decision**, which you can use to show portions of a tutorial even when the Tutorial Manager decision engine determines that a tutorial should not be shown to the current player. If you play a tutorial step associated with a component that respects the off decision, that component only turns on when the Tutorial Manager has decided the tutorial should be shown. If a component does not respect the off decision, then that component turns on whenever the associated tutorial step is played, no matter which decision has been made about showing the tutorial.
- Each tutorial has a limit of 50 steps.
- It is your responsibility to keep track of which tutorials have been completed if your game has more than one tutorial. The Tutorial Manager only keeps track of the most recent tutorial you started.

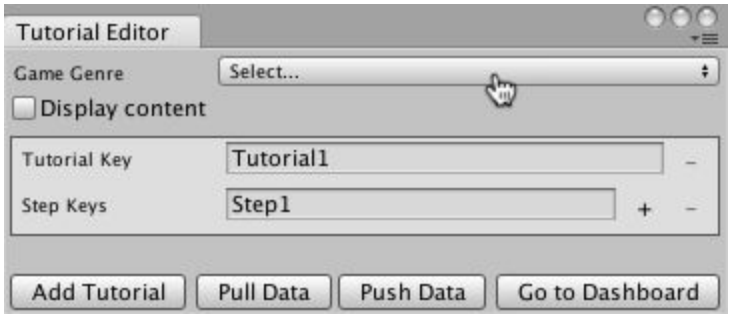
Implementing Adaptive Tutorials

To create an adaptive tutorial, start with one or more normal, non-adaptive tutorials. First, you use the **Tutorial Editor** window to outline the steps in your tutorials. Next, you add adaptive components to those GameObjects in a tutorial that the Tutorial Manager should turn on or off as a player advances through the tutorial.

Creating the Tutorial State Model

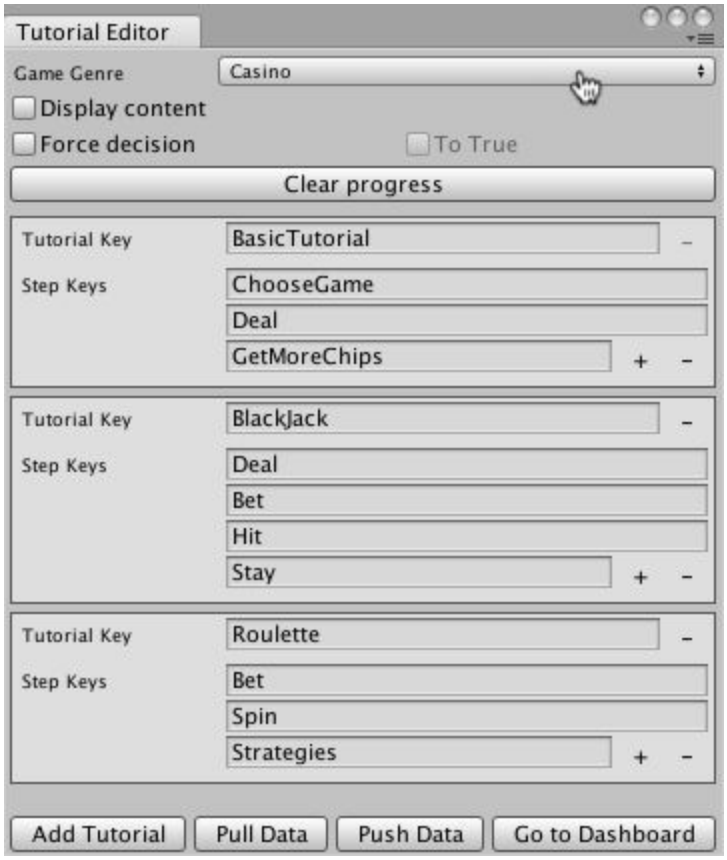
Use the Tutorial Editor window to create a state model that outlines the steps of each of your adaptive tutorials:

1. Open the Tutorial Editor window (menu: **Window** > **Unity Analytics** > **Tutorial Editor**).



2. Set the **Game Genre**. This setting is used by the Tutorial Manager machine learning algorithm. Choose the category that best describes the play style of your game.
3. The Tutorial Manager creates one tutorial entry automatically. If you have more than one tutorial, click the **Add Tutorial** button at the bottom of the window to create additional tutorials.
4. Assign a unique tutorial key name to identify each tutorial.
5. Add steps to the tutorial by clicking the **+** button next to the last existing step. (A single tutorial can contain up to 50 steps.)

6. Assign each step a key name. (The names must be unique within the same tutorial.)



A game with three tutorials, each with 3-4 steps

7. Click **Push Data**. Pushing uploads the structure of your tutorials to the Analytics service and creates or updates the remote text for any AdaptiveText components.

Note: Key names must start with a letter and may contain only letters or numbers.

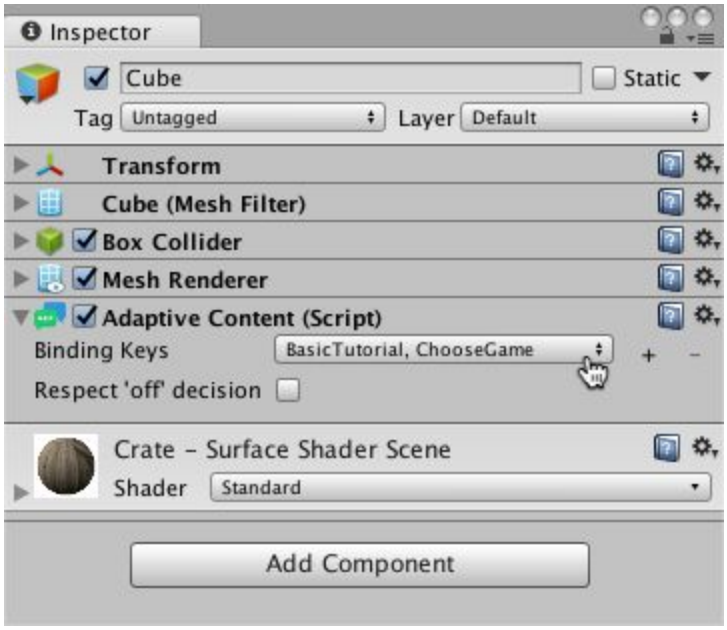
Add Adaptive Components to Your Tutorials

Once you have defined your tutorial outline, you can add the adaptive components that the Tutorial Manager uses to turn the elements of your tutorial on and off as players advance through your tutorial. The Tutorial Manager SDK provides two adaptive components. The AdaptiveContent component turns GameObjects on when a tutorial step begins and turns them off again when the tutorial step finishes. The AdaptiveText component adds the ability to control the text displayed for a tutorial step from the Tutorial Manager dashboard. Put an AdaptiveText component on any tutorial GameObjects that contain a Text, TextMesh, or TextMeshPro object; use AdaptiveContent for other GameObjects.

Note: The Tutorial Manager turns off any GameObjects containing adaptive components when the scene loads (using `GameObject.SetActive(false)`) and turns them back on when the bound tutorial steps are played. Since Unity automatically disables child objects with their parents, avoid nesting adaptive components. (Nesting components can lead to the wrong behavior when you bind them to multiple tutorial steps.)

To add an adaptive component:

1. Select the GameObject in your scene **Hierarchy**.
2. In the **Inspector** window, click **Add Component**.
3. Click **Analytics** in the **Add Component** menu.
4. If the GameObject contains a Text, TextMesh, or TextMeshPro component, select the AdaptiveText component; otherwise, pick the AdaptiveContent component.
5. Bind the component to a tutorial and step using the **Binding Keys** list.
6. To bind a GameObject to more than one tutorial step, click the **+** button next to the list of steps.



A GameObject with an AdaptiveContent component

Playing Tutorials

To play a tutorial, first check whether the Tutorial Manager decision engine recommends that the current player should see a tutorial. If so, then you can start the tutorial using the key name you defined with the Tutorial Editor.

Note: All code that references the Tutorial Manager API needs to import the namespace by calling:

```
using UnityEngine.Analytics;

if (TutorialManager.GetDecision()) {
    TutorialManager.Start("Tutorial1", true);
} else {
    // skip the tutorial
}
```

If you want to present an alternate tutorial instead of skipping a tutorial completely, you can start the alternate tutorial in the `else` clause:

```
if (TutorialManager.GetDecision()) {
    TutorialManager.Start("FullTutorial", true);
} else {
    TutorialManager.Start("ShortTutorial", true);
}
```

When you call `TutorialManager.Start()`, the Tutorial Manager automatically enables any GameObjects containing adaptive components bound to the first step in the tutorial. To advance the tutorial when a player completes a step, call the `TutorialManager.StepComplete()` function. The Tutorial Manager then turns off the adaptive GameObjects associated with the completed step.

What happens next depends on whether you set the `autoAdvance` parameter (the `true` in the examples above) of the `TutorialManager.Start()` to true or false:

- When you set the `autoAdvance` parameter to **true**, the Tutorial Manager advances to the next step in the tutorial by disabling the adaptive components of the previous step and enabling those of the next step.

- When you start a tutorial with `autoAdvance` set to **false**, then the Tutorial Manager does not advance to the next step when you call the `TutorialManager.StepComplete()` function. Instead, the Tutorial Manager disables the adaptive components bound to the previous step and waits for you to call `TutorialManager.StepStart()`.

Note: If the Tutorial Manager cannot contact the Unity Analytics service for any reason and the decision whether or not to show the tutorial has not been previously recorded, the Tutorial Manager shows the default version of the tutorial. (When this occurs, `TutorialManager.GetDecision()` returns true.)

Resuming a Tutorial

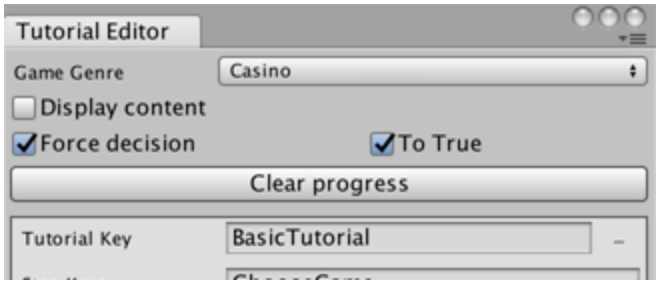
If a tutorial is “paused” or otherwise interrupted before the player completes all the steps, then you can resume the current step of the current tutorial with:

```
TutorialManager.StartStep(TutorialManager.stepId);
```

Note: The Tutorial Manager only keeps track of the current tutorial. If you allow your players to switch between multiple tutorials, then it is your responsibility to keep track of the completion status and step position of each tutorial.

Testing your Adaptive Tutorials

To help you test your tutorials, the Tutorial Editor window includes a **Force decision** option. When you enable this option, Tutorial Manager simulates receiving a decision from the Tutorial Manager service based on the **To True** option.



Tutorial Editor testing options

- When **To True** is checked, then the Tutorial Manager simulates the decision to show the default tutorial and sets its own state accordingly.
- When **To True** is unchecked, Tutorial Manager simulates the decision to not show the default tutorial.

The Tutorial Editor window options only control the Tutorial Manager when you play your game in the Unity Editor. To simulate a decision outside the Unity Editor, the `TutorialManager.GetDecision()` function provides an override that takes a boolean `forceDecision` parameter. When you use this version of *GetDecision()*, you can set the state of the Tutorial Manager state machine to simulate a positive or negative decision from the machine learning algorithm.

The Tutorial Editor window also provides a Clear Progress button. Click this button to reset the Tutorial Manager state. You can also use the `TutorialManager.Reset()` function from code.

Managing your Tutorials

The Tutorial Manager dashboard provides tools for managing the tutorials of your live game.

To open the Tutorial Manager dashboard from the Unity Editor:

1. Load your Unity project in the Editor.
2. Open the Tutorial Editor window (menu: **Window** > **Unity Analytics** > **Tutorial Editor**).
3. Click **Go to Dashboard**.

To open the Tutorial Manager dashboard from a Web browser:

1. Go to https://analytics.cloud.unity3d.com.
2. Log in to your account.
3. Select the project from the project list.
4. Click Tutorial Manager to open the Tutorial Manager dashboard.

Note: Create your tutorials in the Unity Editor and push them to the dashboard using the Tutorial Editor window. Until you push a tutorial from the Editor, the dashboard has nothing to display.

Pushing and Pulling Tutorial Data

You can push and pull your tutorial model and text using the Tutorial Editor window in Unity.

Pushing from your project in the Unity Editor overwrites the tutorial structure and adaptive text stored in the Analytics service. The data is pushed to the **Development** environment, overwriting any existing data in that environment.

Pulling to your project updates the text of any AdaptiveText components to reflect the **Development** version of the text fields on the Tutorial Manager dashboard.

To push or pull your tutorial data:

1. Open the Tutorial Editor window in Unity (menu: **Window** > **Unity Analytics** > **Tutorial Editor**).
2. At the bottom of the window, click **Pull Data** or **Push Data**.

Using Tutorial Manager with version control software

If you use Tutorial Manager in a shared development environment with other creators, you probably use some form of version control software, such as git, mercurial, or Unity Contribute. In order to preserve your tutorial content between contributors, the Tutorial Manager creates a data file which can be committed to your repository. This file lives within your project at the following location:

```
/Assets/Resources/unity_tutorial_manager.dat
```

This file contains the data you enter in the Tutorial Editor, such as tutorial and step IDs.

Release and Development environments

The Tutorial Manager service maintains two separate environments. Tutorial Manager code running in the Unity Editor pushes and pulls data from the Development environment. Live games receive data from the Release environment. You can copy tutorial data from one environment to the other on the Tutorial Manager dashboard.

Tutorial Manager environments

This separation of environments allows you to improve and test your tutorials without affecting your live players until your changes are ready to publish.

Note: Whenever you change the data in the Release environment, whether by copying your data from Development or by editing tutorial text directly in the Release environment, the Tutorial Manager resets the running experiment.

Managing your Adaptive Text

The Tutorial Manager automatically creates a remote text field for your AdaptiveText components. These remote fields are updated when you push data from your project to the Analytics service. If you make changes to the text on the Analytics dashboard, you can pull those changes to your project. At runtime, the current text of the Remote Settings takes precedence over the text in the project. You can push and pull data from the Tutorial Editor window in Unity (menu: **Window** > **Unity Analytics** > **Tutorial Editor**).

Tutorial Manager provides both a **Release** and a **Development** environment. When you play your game in the Unity Editor and in development builds, the AdaptiveText component uses the text in the Development environment. When you run a release build, the component uses the text from the Release environment. (You can choose which type of build to create using the **Development Build** checkbox on the **Build Settings** window in the Editor.) If the Tutorial Manager cannot contact the Analytics service at runtime and no cached model exists, then the text of adaptive components is not changed.

Note that the AdaptiveText component has a property, **Ignore if remote is empty**, that you can set in the Inspector window. When you set **Ignore if remote is empty** to true, the text associated with the content is not updated when the corresponding remote value is an empty string. If **Ignore if remote is empty** is false, then the text is replaced by an empty string. The **Ignore if remote is empty** property applies both at runtime and when you pull data to your project using the Tutorial Editor window **Pull Data** button.

Tutorial Reports

The Tutorial Manager dashboard provides two reports.

The Retention report compares retention results between a control group and test group (the Tutorial Manager group). Players in the control group always see the default tutorial (essentially, `GetDecision()` always returns true). Players in the test group receive a decision made by the Tutorial Manager machine learning algorithm. Some players will see the default tutorial (`GetDecision()` returns true); other players will see no tutorial — or your alternate tutorial, depending on your implementation (`GetDecision()` returns false).

DAY 1 RETENTION

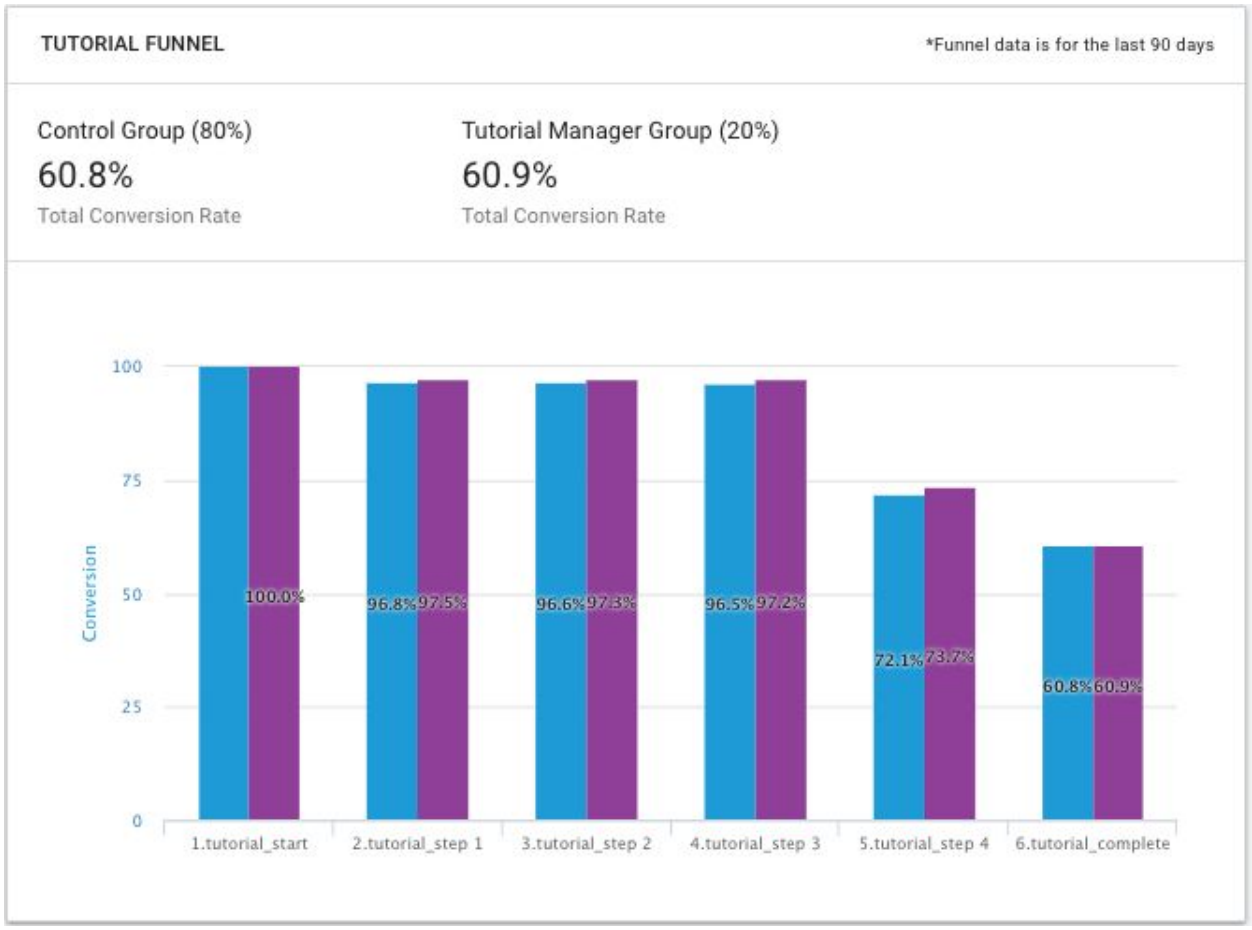
05/15/2018 - 06/14/2018

Control Group (80%)	Tutorial Manager Group (20%)
31%	32% +1%
Day 1 Retention	Day 1 Retention

Metrics	Control Group (80%)	Tutorial Manager Group (20%)	Result
New Users	14,374	3,701	18,075
Retention			
Day 1 Retention	31%	32%	+1%
Day 7 Retention	11%	7%	-4%
Day 30 Retention	14%	0%	-14%
Sessions			
Total Sessions	62,576	16,011	
Average Session Length	6.8 min	7 min	+3%

The Retention report

For each tutorial that you create using the Tutorial Editor window, the Tutorial Manager dashboard automatically creates a funnel report. A funnel displays the percentage of player progress from one step to the next. The Tutorial Manager funnel report compares the conversion of players in the control group to that of players in the test (Tutorial Manager) group.



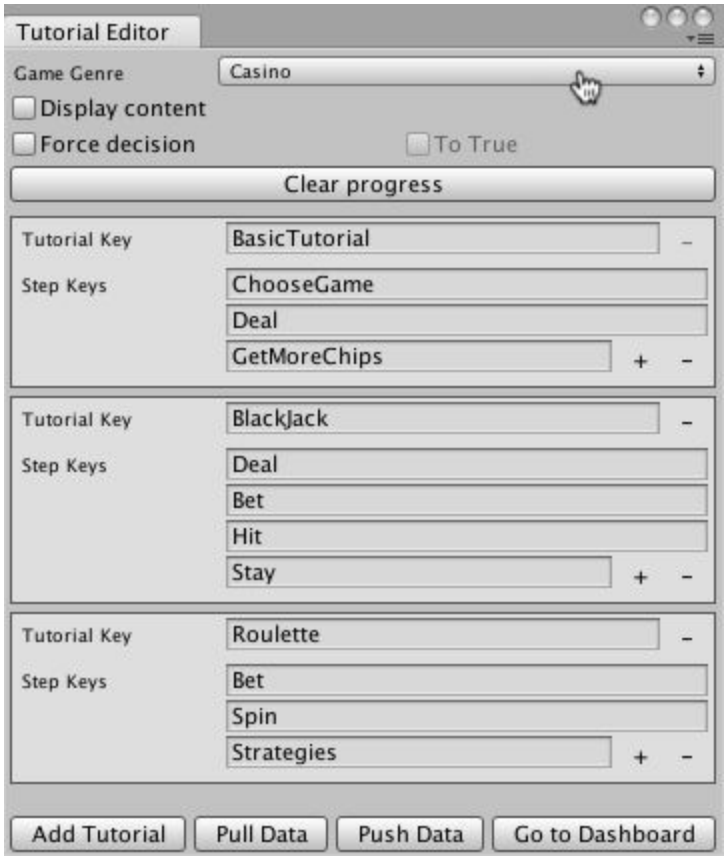
The tutorial funnel

Note: If you set up a tutorial so that it is only played when `TutorialManager.GetDecision()` returns false, then players in the control group will never see that tutorial. Thus the funnel reports will show zero players from the control group for such tutorials.

Tutorial Manager Reference

Components and Inspectors

Tutorial Editor Window



Element	Purpose
Game Genre	The genre of your game. Choose the genre that best describes your game.
Display content	When checked, the editor window displays the text of AdaptiveText components beneath the associated tutorial step.
Force decision	Simulates a specific decision from the Tutorial Manager decision engine while testing in play mode in the Unity Editor. Set the decision to simulate with the **To True** option.
To True	Check the **To True** option to simulate a decision to show the tutorial;

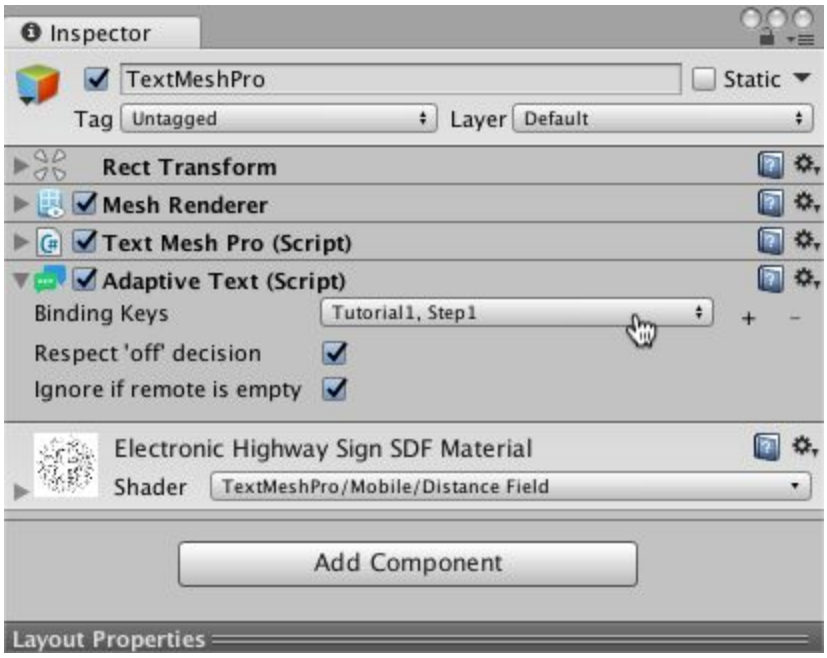
	uncheck the box to simulate a decision to not show the tutorial.
Clear progress	Clears the Tutorial Manager state.
Tutorial Section	Each tutorial has its own section in the window.
Tutorial Key	A unique identifier for the tutorial.
Delete Tutorial Button	Click **-** to delete the tutorial. (You cannot delete the first tutorial in the window.)
Step Keys	A unique identifier for each step in the tutorial.
Step Text	(Read only) The text associated with the tutorial step. Shown when **Display content** is checked.
Add, Remove Step Buttons	Click the **+** or **-** buttons next to the last step to add or remove steps at the end of the tutorial. Each tutorial must have a minimum of one step.
Add Tutorial	Add a new tutorial section to the game.
Pull Data	Pulls data from the Development environment of the Tutorial Manager service. **Caution:** Pulling data overwrites all changes you have made to the tutorial structure and adaptive text since the last time you pushed your data to the service.
Push Data	Pushes data to the Development environment of the Tutorial Manager service. **Caution**: Pushing data overwrites the tutorial structure and adaptive text currently in the Development environment.
Go to Dashboard	Opens the Tutorial Manager dashboard.

Adaptive Content Component

The AdaptiveContent component turns on its parent GameObject when the Tutorial Manager starts one of the tutorial steps to which it is bound and turns it off when the step is complete.

Element	**Purpose**
Binding Keys	Each key binds the AdaptiveComponent to a tutorial and step. (Enter the keys for your tutorials and steps on the Tutorial Editor window.)
Binding +, - buttons	Click **+** to add bind the component to an additional tutorial step. Click **-** to remove a binding. Each component must have a minimum of one binding.
Respect 'off' decision	When checked (true), the associated GameObjects are only enabled when the Tutorial Manager has decided the default tutorial should be shown (`TutorialManager.GetDecision()` returns true). When unchecked (false), the associated GameObjects are enabled whenever the tutorial step is started by the Tutorial Manager.

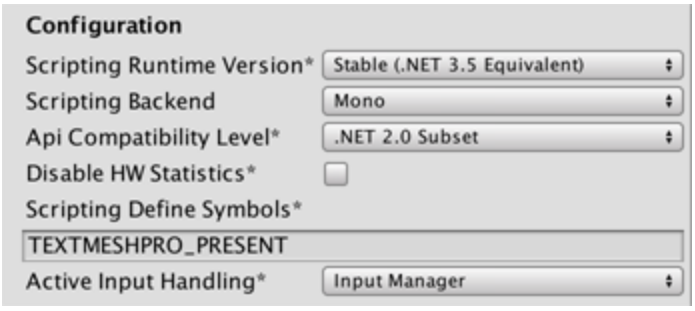
Adaptive Text Component



The AdaptiveText component extends the AdaptiveContent class. In addition to controlling whether its GameObject is active, the AdaptiveText component also controls the text of a Text, Text Mesh, or Text Mesh Pro component attached to the same GameObject.

Important: To use Tutorial Manager with TextMeshPro, you must add the `TEXTMESHPRO_PRESENT` scripting define symbol:

1. Open the Player Settings window (menu: **Edit** > **Project Settings** > **Player**).
2. Click to open the **Other Settings** section, if necessary.
3. Under Configuration, enter `TEXTMESHPRO_PRESENT` into the Scripting Define Symbols text box.



When you add an AdaptiveText component to a GameObject, bind it to a tutorial step, and push your tutorial state model to the Unity Analytics service, the Tutorial Manager creates a remote text field on the Tutorial Manager dashboard. At runtime, the value of the remote field takes

precedence over the value entered in the Unity Editor. Use the **Push** and **Pull Data** buttons on the Tutorial Editor window to keep your project in sync with your dashboard.

Only bind a single AdaptiveText component to any individual tutorial step. (If you bind multiple AdaptiveText components to a step, the text of all of these components is updated to the value of the same remote text field.) If you need more than one text component for a step, you can use AdaptiveContent components instead. However, only the text associated with the AdaptiveText component can be updated remotely from the Tutorial Manager background.

Element	**Purpose**
Binding Keys	Each key binds the AdaptiveComponent to a tutorial and step. (Enter the keys for your tutorials and steps on the Tutorial Editor window.)
Binding +, - buttons	Click **+** to add bind the component to an additional tutorial step. Click **-** to remove a binding. Each component must have a minimum of one binding.
Respect 'off' decision	When checked (true), the associated text objects are only enabled when the Tutorial Manager has decided the default tutorial should be shown (`TutorialManager.GetDecision()` returns true). When unchecked (false), the associated text objects are enabled whenever the tutorial step is started by the Tutorial Manager.
Ignore if remote is empty	When checked, the Tutorial Manager does not update the associated text component if the remote field contains an empty string. When unchecked, entering an empty string into the remote field removes the text from the text component. The **Ignore if remote is empty** setting applies both at runtime and when you pull data from the service.

API

The public API of the Tutorial Manager consists of a single class, TutorialManager, which is defined in the `UnityEngine.Analytics` namespace.

TutorialManager Class

The TutorialManager class provides the interface for accessing and controlling the Tutorial Manager state machine. All the properties and methods are static, so you do not need to construct an instance of the TutorialManager class.

Class Summary

public static bool autoAdvance { get; }
public static bool complete { get; }
public static bool showTutorial { get; }
public static string stepId { get; }
public static int stepIndex { get; }
public static string tutorialId { get; }
public static int tutorialLength { get; }

public static bool GetDecision();
public static bool GetDecision(bool forceDecision);
public static string GetStepIdAtIndex(int index);
public static bool Start(string tutorialId, bool autoAdvance = true);
public static void Skip();
public static void StepStart(string stepId = null);
public static void StepComplete();
public static void Reset();

Properties

autoAdvance

[Read Only] A flag indicating whether the tutorial should automatically progress from one step to the next. By default, the Tutorial Manager will progress from one step to the next automatically. This is often desirable, but not every tutorial works this way. If your game's tutorial has steps interspersed with gameplay, you might want more precise control. By calling `Start(tutorialId, false)`, this flag will be set to `false`. The result is that calling `StepComplete()` will complete a step, but not automatically start the next one. This allows gameplay to proceed with the tutorial "paused" between states. You will need to call `StepStart()` to start the next step.

`true` if auto advance; otherwise, `false`.

Declaration:
```
public static bool autoAdvance { get; }
```

complete

[Read Only] A flag indicating whether the current tutorial is complete.

`true` if the current tutorial is marked as complete; otherwise, `false`.

Declaration:
```
public static bool complete { get; }
```

showTutorial

[Read Only] Reflects the server recommendation as to whether the player should see the tutorial.

`true` if the tutorial should be shown; otherwise, `false`.

Declaration:
```
public static bool showTutorial { get; }
```

stepId

[Read Only] The binding ID of the current tutorial step.

A string representing the binding ID of the current step.

Declaration:
```
public static string stepId { get; }
```

stepIndex

[Read Only] The index of the step in the currently running tutorial.

The index of the step.

Declaration:
```
public static int stepIndex { get; }
```

tutorialId

[Read Only] The ID of the current tutorial, as set by `TutorialManager.Start(tutorialId)`

The tutorial identifier.

Declaration:
```
public static string tutorialId { get; }
```

tutorialLength

[ReadOnly] The number of steps in the current tutorial.

The length of the tutorial.

Declaration:
`public static int tutorialLength { get; }`

Methods

GetDecision()

Retrieves the show/no show recommendation from the Tutorial Manager Server.

This method is the lynchpin of the TutorialManager system. It should be called when the player arrives at the first tutorial. Calling it lets you know what recommendation the server has for this player, and informs the server that this recommendation has been acted upon.

If you need to know the decision at any point other than the moment that the player arrives at the decision point, use `showTutorial`.

```
if (TutorialManager.GetDecision()) {
    TutorialManager.Start("Tutorial1");
    // Any other code required by your tutorial
} else {
    // skip the tutorial
}
```

Returns `true`, if the tutorial should be shown, `false` otherwise.

Declaration:
`public static bool GetDecision()`

GetDecision(bool forceDecision)

Mimics the show/no show recommendation from the Tutorial Manager Server, but overrides with a specific decision.

This version takes a boolean `forceDecision` parameter, which forces either `true` or `false`. This is useful for testing and QA, when you want to ensure a specific outcome. To use the server recommendation, call `GetDecision()`.

```
if (TutorialManager.GetDecision(true)) {
    TutorialManager.Start("Tutorial1");
    // Any other code required by your tutorial
} else {
```

```
        // skip the tutorial
    }
```

Note that when you enable the **Force decision** option on the Tutorial Editor window, then this version of the GetDecision() function is always called when playing from the Unity Editor. The decision uses the **To True** setting of the Tutorial Editor window.

Returns `true`, if `forceDecision` is `true`, `false` otherwise.

Declaration:
```
public static bool GetDecision(bool forceDecision)
```

GetStepIdAtIndex(int index)

Returns the string id of a step at the specified index.

Declaration:
```
public static string GetStepIdAtIndex(int index)
```

Start(string tutorialId, bool autoAdvance = true)

Starts a tutorial.

```
    if (TutorialManager.GetDecision()) {
        TutorialManager.Start("Tutorial1");
    } else {
        // skip the tutorial
    }
```

- **tutorialId**: The binding id representing the current tutorial
- **autoAdvance**: If 'true' (default) ending one tutorial step will automatically advance to the next step.

Returns `true`, if tutorial should be shown, `false` otherwise.

Declaration:
```
public static bool Start(string tutorialId, bool autoAdvance = true)
```

Skip()

Call this if the player opts to skip the tutorial (in the case you have a skip option in your game).

This will clear the current tutorial state

Declaration:
```
public static void Skip()
```

StepStart(string stepId = null)

Call this each time the player begins a step in the tutorial.

It is unnecessary to call this method when `autoAdvance` is set to `true`. Under that condition, `StepComplete()` will start the next step automatically. If you provide a specific `stepId`, the named step will start.

- **stepId**: (optional) The step of the current tutorial to start.

Declaration:
```
public static void StepStart(string stepId = null)
```

StepStart(string tutorialId, string stepId = null, bool autoAdvance = true)

Call this to start a step in some tutorial other than the current one.

- **tutorialId**: The tutorial to go to.
- **stepId**: The step of the new tutorial to start.
- **autoAdvance**: If `true` (default) ending one tutorial step will automatically advance to the next step.

Declaration:
```
public static void StepStart(string tutorialId, string stepId, bool autoAdvance = false)
```

StepComplete()

Call this each time the player completes a step in the tutorial.

Declaration:
```
public static void StepComplete()
```

Reset()

Clears the runtime Tutorial Manager state.

This method is provided for QA and testing purposes only. Calling it will null out all prior decisions and runtime progress on the current device.

Declaration:

```
public static void Reset()
```