



Welcome to the articy importer plugin. In this document we will give you a quickstart on how to use the articy Unity plugin, give you a broad overview of all the features the plugin has to offer and dive into some more advanced features.

A full API documentation and guides for more specific topics can be found here:

➤ <https://www.articy.com/articy-importer/unity/>

If have any questions or want to report a bug, you can contact the support via support@articy.com

Table of contents

1. Introduction.....	2
2. Features.....	2
3. Platform support	2
4. Quick Start	3
5. Articy Database	5
6. Articy Flow Player	5
7. Scripting.....	6

1.Introduction

The purpose of the plugin is to give you a quick and easy way to access your articy:draft data in Unity. The plugin automatically parses the data exported from articy:draft using the built-in Unity exporter. All your namespaces and variables are converted to C#, making it extremely easy to use them directly inside your scripts.

Apart from simply giving you access to the database, the articy Unity plugin also includes powerful tools that simplify using your data within a game. To play back flows or branching dialogues, check out the [ArticyFlowPlayer](#). These tools are introduced in more detail in ↗ [Dialogues and Flow Traversal](#)

2.Features

- Simple C# API to access your articy:draft data
- Flow player for automatic configurable flow traversal with automatic script evaluation
- Fast evaluation of articy:drafts script language(articy:expresso)
- Built-in localization

3.Platform support

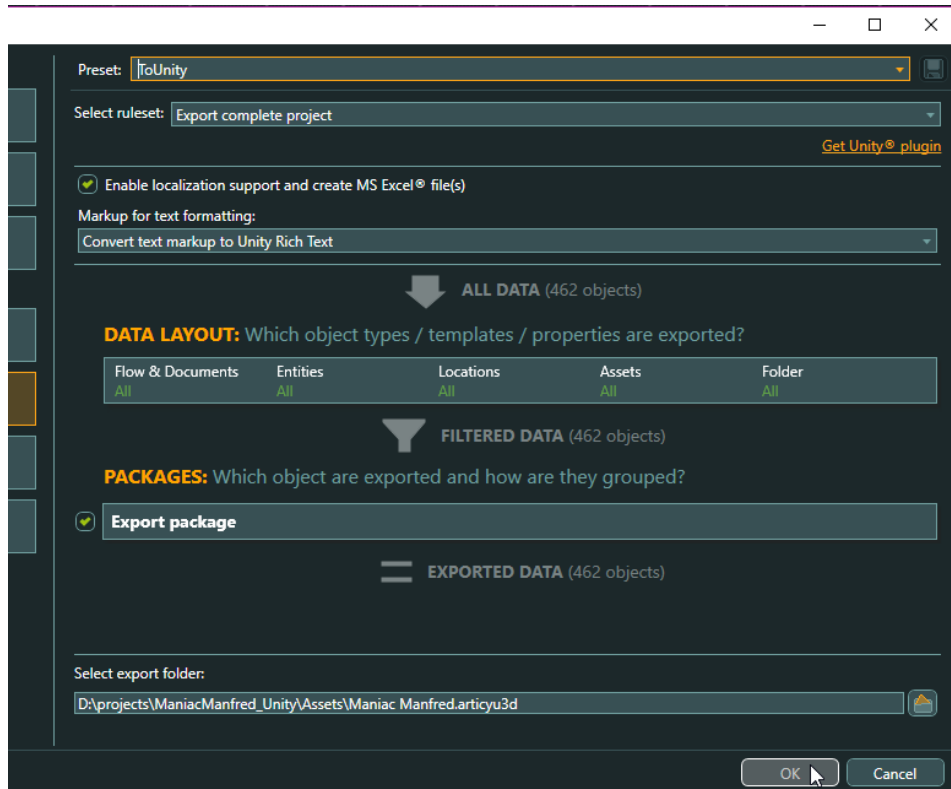
The plugin was built to support all unity supported platforms. That means no native code was used and the runtime only uses unity references.

We successfully tested the runtime part of the plugin on the following platforms:

- Windows
- UWP
- Linux
- Mac
- iOS
- Android
- WebGL

4. Quick Start

Export your articy:draft project using the *Unity export*. Choose your Unity project asset folder as the target folder under "Select export folder".



Tip

You can save an articy:draft export preset, so you don't have to choose the unity asset folder every time, instead just select the preset and hit export.

Once exported, switch back to Unity. The plugin will automatically start importing the newly exported file. Once the plugin is finished and the progress bar has disappeared, your articy:draft data has been successfully imported.

The easiest way to access any data from the articy database is by using an *ArticyRef* in your scripts. Create a new *GameObject* in your scene. Add a new script, call it **MyArticyPluginComponent** and add it to your new *GameObject* as a component. Write the following code for your newly created script component:

C#

```
using UnityEngine;
using Articy.Unity;

public class MyArticyPluginComponent : MonoBehaviour
{
    public ArticyRef myFirstArticyModel;

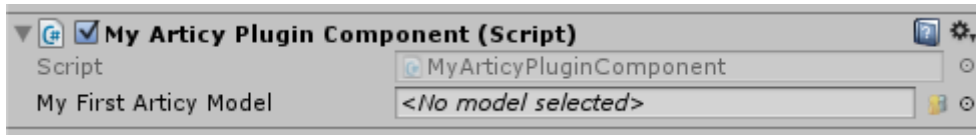
    void Start()
```

```

{
    var techName = myFirstArticyModel.GetObject().TechnicalName;
    Debug.Log (techName);
}

```

Save your code file and head back to Unity. Select your previously created GameObject and check the Inspector Window. It should look something like this:



An [ArticyRef](#) is used to store references to your articy data. This data can be of any type, from individual entities, to dialogues or flows. To select what object you want to reference, click the little browse button to the far right of the field. This will open the articy:draft model picker. Choose one of your models by double-clicking it. The chosen model is now referenced by your script component and can be accessed, like we did in the Start() method in the example above. To access different properties and template data of your objects, please see [Object handling with ArticyRef](#).

Note

If no objects are shown in the object picker, make sure that you have created content in your articy:draft project and that you have properly configured your export definition and packages.

Now if you hit the play button in the Unity Editor, you should see the technical name of the object printed into the unity log window. Congratulations you have made your first step in using the articy Unity plugin!

5. Articy Database

While the beforementioned ↗ [ArticyRef](#) is a convenient way to reference and access articy data in your scripts, you can also query the ↗ [ArticyDatabase](#) directly. With the ArticyDatabase you can query objects by their ↗ [object id](#), ↗ [technical name](#) or by ↗ [type](#). The ArticyDatabase also handles managing your packages and hosts relevant objects for script execution.

Important

It is worth noting that you don't have to create, setup or unload the database. It will automatically be created when your articy:draft data is imported and is available via its static methods from anywhere.

6. Articy Flow Player

Accessing your data is only one aspect of the articy unity plugin. Another core feature is the ↗ [ArticyFlowPlayer](#) component. The ArticyFlowPlayer is an easy to use flow traversal engine, that can be customized to your needs.

In a nutshell: The flow player will traverse a given flow the way it was set up in articy:draft. It can evaluate conditions, execute instructions, handle branching and will make callbacks into your game code - so your game can display dialog options for example.

To use the flow player, just attach it to one of your game objects, set the desired settings, implement the ↗ [IArticyFlowPlayerCallbacks](#) interface on one of your scripts on the same game object and react and manipulate the flow player to your liking.

If you want to know more about the articy flow player, see the ↗ [Dialogues and Flow Traversal](#).

7.Scripting

This section contains the following subsections:

- [Global Variables](#)
- [Scripts](#)
- [Script Methods](#)

Global Variables

As part of your imported data, the plugin will also create the GlobalVariables structure from your articy:draft project, including all of your variable sets. It is highly recommended to make use of these global variables sets, since the plugin can use them automatically when evaluating branches and modify them inside instructions. You can access your variables from your scripts via the ArticyGlobalVariables object:

ArticyGlobalVariables.Default.<Your variable set>.<Your variable>

You could also utilize the methods found on ↗ [BaseGlobalVariables](#) to get more advanced access to your variables. Also found on BaseGlobalVariables is the ↗ [ArticyNotificationManager](#), which can be used to add listener methods to get informed when a specific or multiple variables are modified. It's also possible to create additional copies of your global variables sets. The default set of global variables is stored inside ↗ [DefaultGlobalVariables](#). For the most use cases the default one is enough and will automatically be used by all **ArticyFlowPlayers**. But you can explicitly tell a flow player to work with a different copy of the variables instead, by assigning its globalVariables property

Scripts

All your articy:draft scripts found on pins, conditions, instructions and inside your templates are converted to C# code, making script execution extremely fast and convenient to use directly from C#.

Scripts inside flows and dialogues will automatically be executed by the ArticyFlowPlayer while traversing Scripts inside your templates can be called on demand by using ↗ [CallScript\(\) \(Condition\)](#) or ↗ [CallScript\(\) \(Instruction\)](#) when you want to execute them.

Caution

If you have previously used articy:access, it's important to understand that not all of the supported expresso script features are supported by the plugin!

Script Methods

Scripts inside articy:draft can not only be used to check or modify global variables. They also allow you to call arbitrary methods. While of no use inside articy:draft those methods

are much more important inside your Unity project. The import process will generate an interface called `IScriptMethodProvider`. This interface contains all your methods found inside your articy scripts. You just need to implement this interface on one of your script components on the same GameObject hosting the `ArticyFlowPlayer`, and the flow player will make sure to call your method implementations while traversing the flow.

Caution

When implementing methods, you have to check the circumstances under which your method is called. The flow player will use something called *Forecasting* which is basically the processors way of looking into the future to test if upcoming paths and branches are valid. The important part about forecasting is, that the flow player might temporarily change variables to ensure correct script behavior when testing a branch, and will automatically and transparently revert those forecasting variable changes. While this is easy to maintain internally for variables, it is not so easy for methods, especially for user code that could be called and can basically do anything. So it is the users responsibility to take care of when his methods are called and if and what they can and should change. To check if the flow player is just testing or if the method is actually called you can use ↗ [IsCalledInForecast](#) on your `IScriptMethodProvider` implementing class. The most common implementation of methods returns immediately if it finds that variable to be true.
