Figma Converter for Unity

manual for developers

5.4.1

Introduction

I strongly recommend reading
this manual before using the asset.

1. This asset can work in conjunction with other assets using their capabilities.
The currently supported assets can be found under the "**DEPENDENCIES**" tab on the asset's page in the Asset Store.
In order to work with these assets, you need to **buy** them from the **Asset Store** and **import** them **into** your **project**.
After you have done this, carefully **read** both **manuals** - for developers and for designers.
In the contents of these manuals, you will find **page numbers** for information on the **use** of these **assets** and their corresponding **tags**, if any.

2. If you encounter any errors while working with the asset, please write me about it at provided contacts.
I typically respond quickly to messages, offer assistance on an individual basis, and address any identified bugs in the upcoming updates.
You can leave comments about the features that you want to see in the asset - it's will also be considered.

   Discord Server: https://discord.com/invite/ZsnDffV5eE

   Telegram Group: https://t.me/da_assets_publisher

   Email Support: da.assets.publisher@gmail.com

   Website: https://da-assets.github.io/site/

3. Usually, **to reproduce your issue, I need access to your project in Figma.**
If you are working in a company, you might need to coordinate granting Figma project access with your management.
I follow a confidentiality policy, your project will not be used for any purposes other than assisting with your issue.

4. Information about changes in the manual can be found in the changelog available on the developer's website.

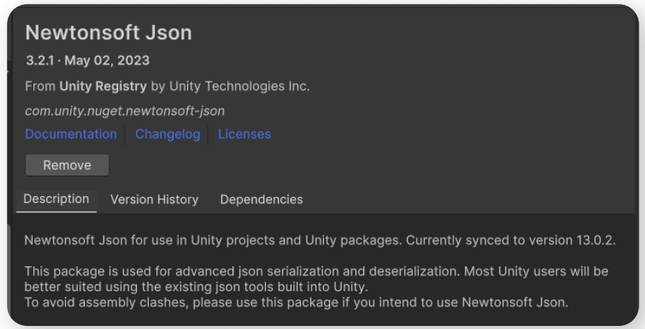
5. If you see any mistakes in the manual, or oddities or bugs in the operation of the asset, please report it to developer using known contacts.

6. You can earn a percentage from sales of my assets through the Unity Affiliate Program.
If this interests you, please contact me via PM or email.

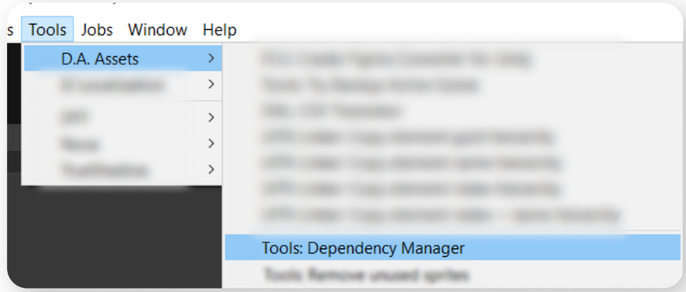
Contents

Json.NET

1. The asset requires the "**com.unity.nuget.newtonsoft-json**" package (**Json.NET**) to function.
If you download **Json.NET** from another source or use the built-in Unity version, the asset is likely not going to work.



2. After installing Json.NET through the Package Manager, check if Figma Converter recognizes this dependency.
Open the asset's "**Dependency Manager**" through the context menu.

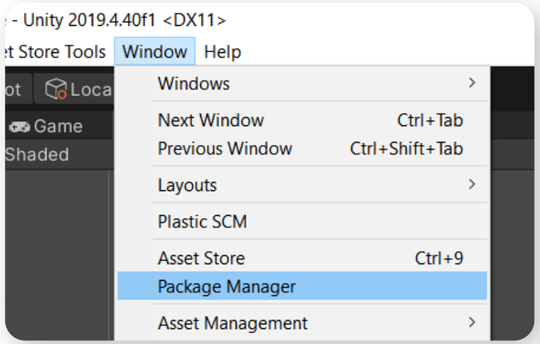


3. If Json.NET is installed correctly, you will see that this dependency is marked as "**ENABLED**."

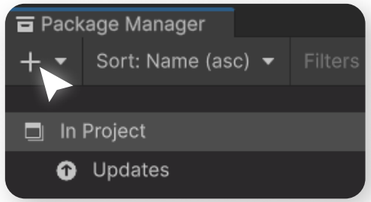
In the slides below, you will find instructions for installing **Json.NET** using the Package Manager.

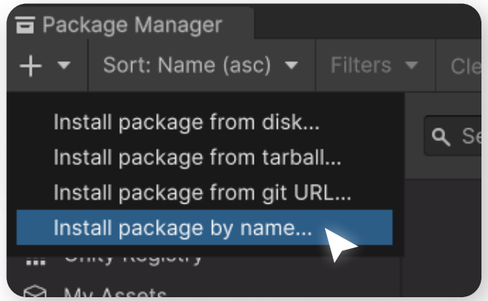
Json.NET

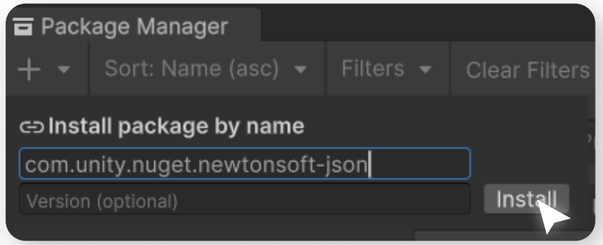
3 To install "**Json.NET**", open the Unity Package Manager.



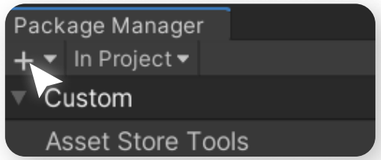
4 Click on the "**+**" button, and then, in the menu that appears,
click on "**Install package by name**" menu item.
Enter the package name "**com.unity.nuget.newtonsoft-json**" and click on the "Install" button.

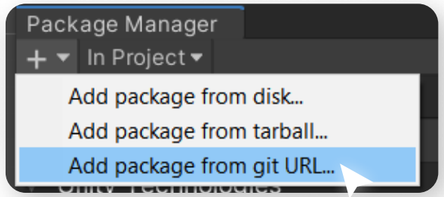


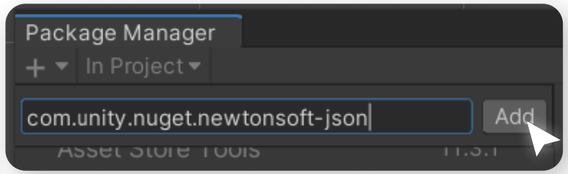




5 As an alternative, you can use the "**Install package from git URL**" function.





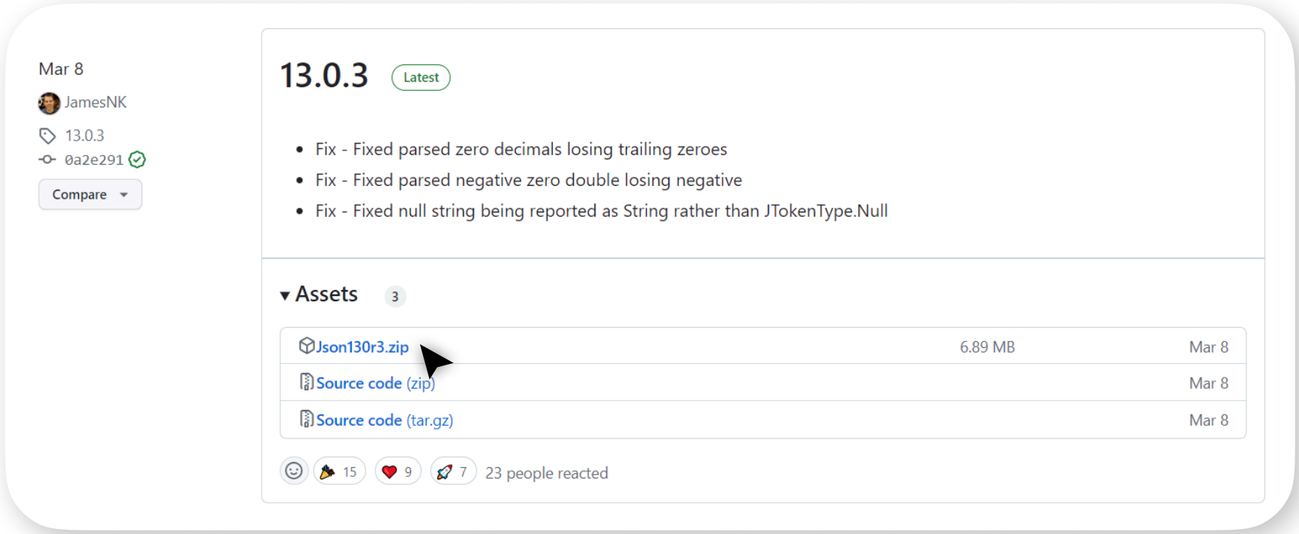


After installing Json.NET, you can continue using the asset.

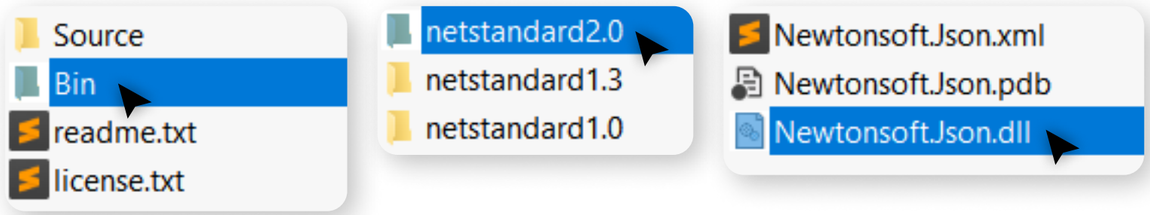
Json.NET

6 If the installation of **Json.NET** through the Package Manager was unsuccessful, download the latest release of **Json.NET** from the official repository: https://github.com/JamesNK/Newtonsoft.Json/releases

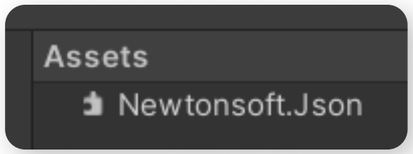
Do not use **Json.NET** versions released before 2020.



7 **Unzip the archive**, open the **Bin** folder, then **netstandard2.0**, and drag the **Newtonsoft.Json.dll** into the **Assets** folder in your project.

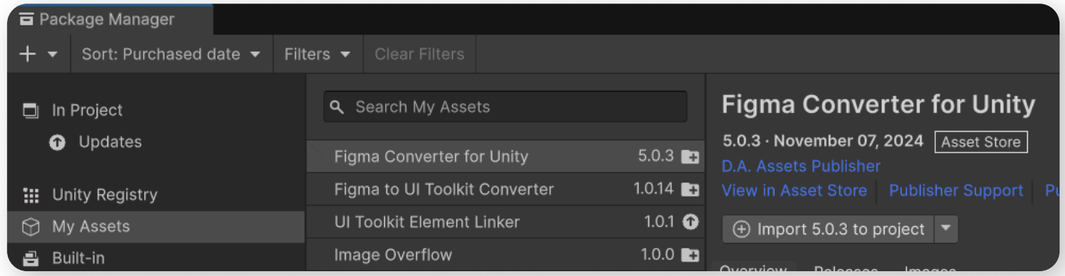


8 After installing Json.NET, you can import the Figma Converter for Unity using the Package Manager.

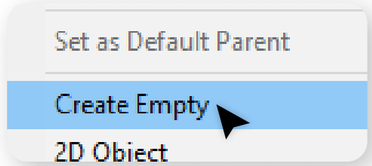


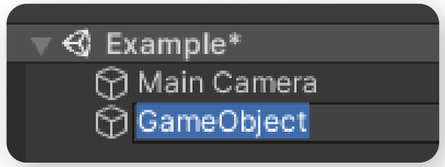
Scene Setup

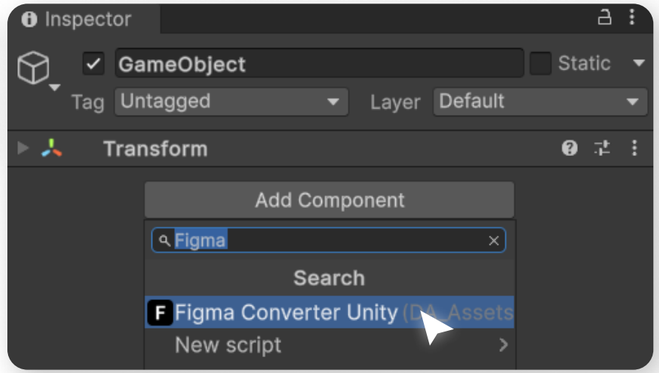
1. Import the "**Figma Converter for Unity**" asset using the Unity Package Manager.



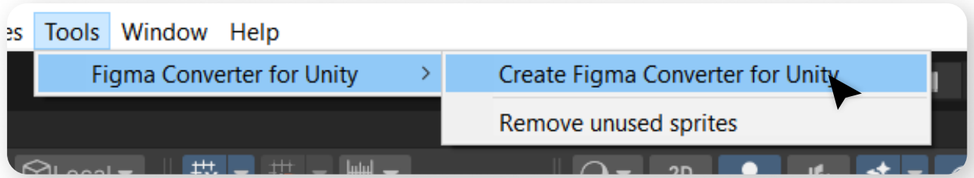
2. Create an empty GameObject on the scene, and then add the "**FigmaConverterUnity**" script on it.





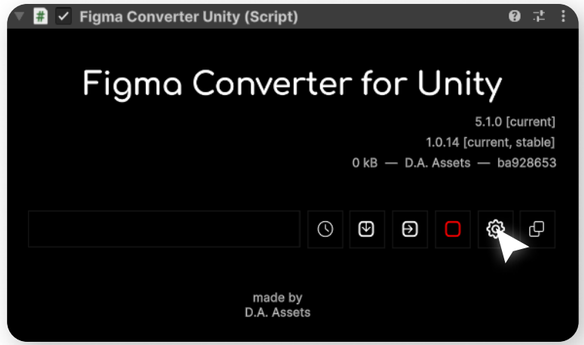


3. Also, you can create an asset in the scene using the menu.

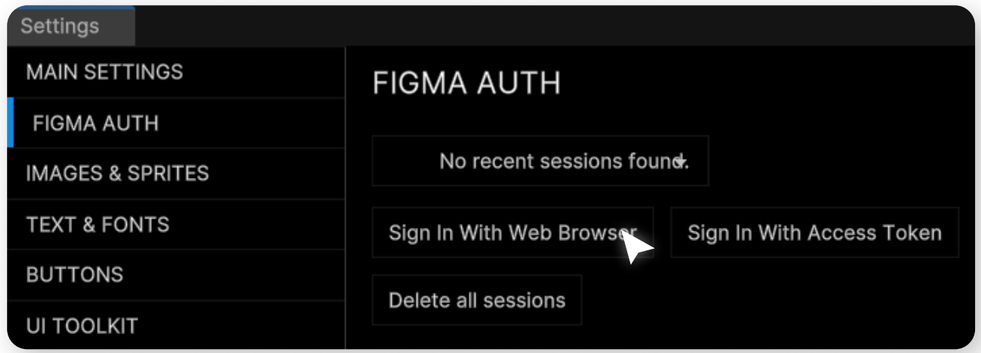


Auth

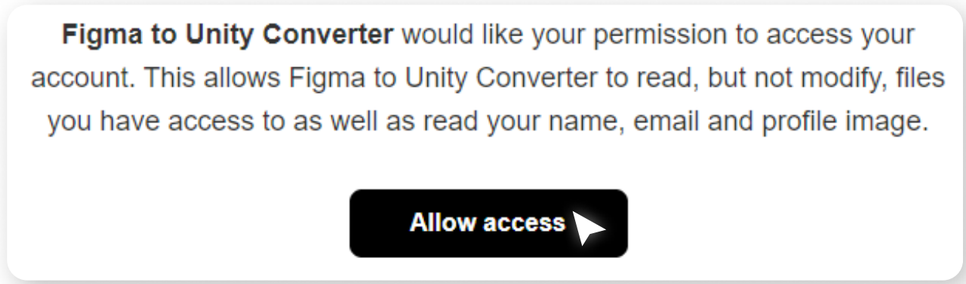
1. Now, you need to log in to your Figma account inside the asset.
To do this, **open** the asset's **settings.**



1. Then open "**FIGMA AUTH**" tab and press "**Sign In With Web Browser**" button.

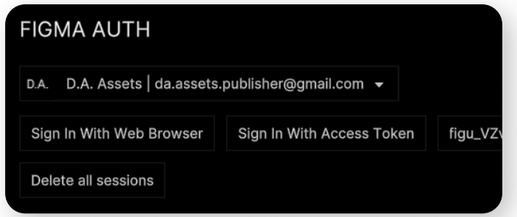


2. In the browser that opens, click on the "**Allow access**" button.

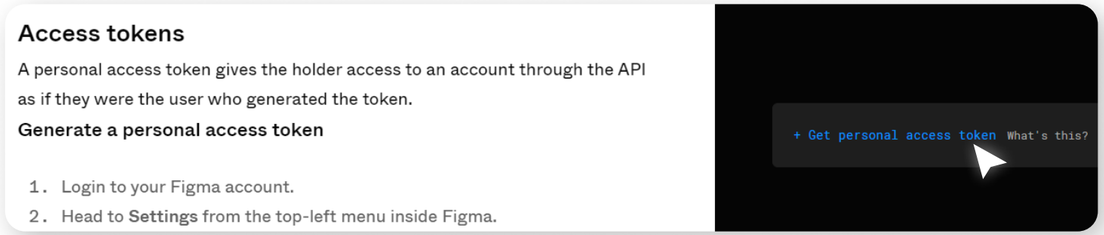


Auth

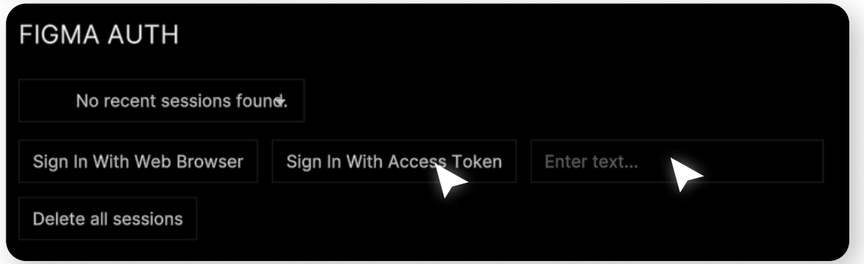
3 After that, under the logo you will see the name of your **authorized account** - this means that the authorization was successful, and now you can proceed with the import.



4 If for some reason you are unable to obtain the token using the asset, you can get it on the Figma website.
To do this, follow this link: https://www.figma.com/developers/api#access-tokens
Please check if you are logged in to this website with the Figma account that has access to the project you want to import.
To obtain the token, click on the "**Get personal access token**" button.



5 Then, copy the obtained value and paste it into the "**Token**" field, then click on the "**Sign In With Access Token**" button.



6 After this, authentication will occur based on the entered token, and you will see a message in the console.

Auth

7 If you do not want to receive a token manually and when you try to receive a token using an asset, you see the error "**SocketException: An attempt was made to access a socket in a way forbidden by its access permissions**", you can use the solution suggested by one of the users of the asset.

The author of the asset has not tested this solution and **not responsible for the consequences of its use**.

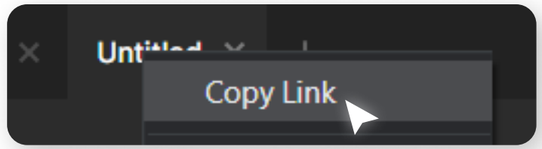
Steps (for Windows):

1. Open CMD.exe as administrator and type "net stop winnat", then press Enter;
2. Type "netsh int ipv4 add excludedportrange protocol=tcp startport=1923 numberofports=1", then press Enter;
3. Type "net start winnat", then press Enter;
4. Try auth in the asset again.

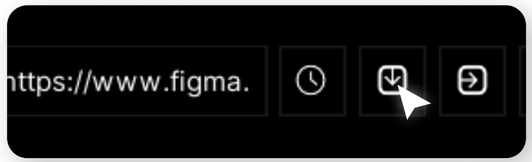
Import Frames

Before importing a layout, check **project permissions** for editing (see the **Teamwork** section in the **Manual for designers**).

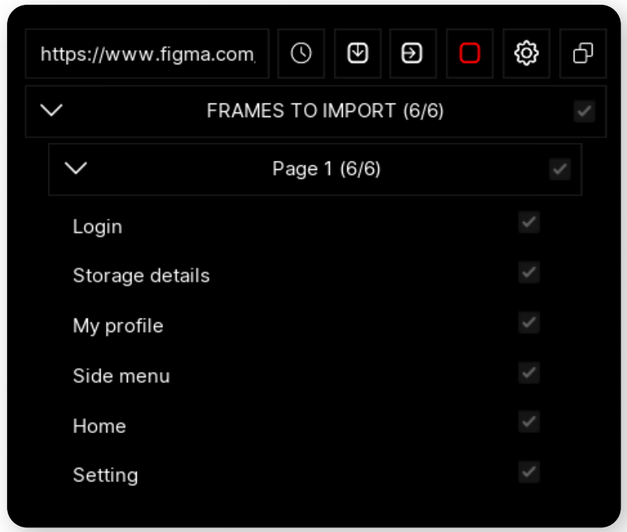
1. Open the figma project you are about to import and get a link to it. It can be obtained by **right-clicking on the tab** with an open project.
Example link: https://www.figma.com/file/XXXXXXXXXXXXXXXXX...



2. Press "**Download**" button to download your project and get a list of its pages and frames.



3. After the project has downloaded, you can select the pages and frames you want to import.

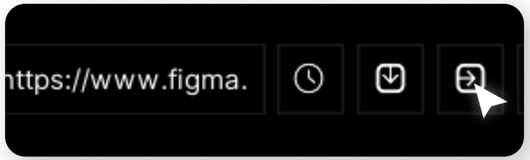


To have the components you want to import appear in the "**FRAMES TO IMPORT**" list, you must place them in a **Frame**.

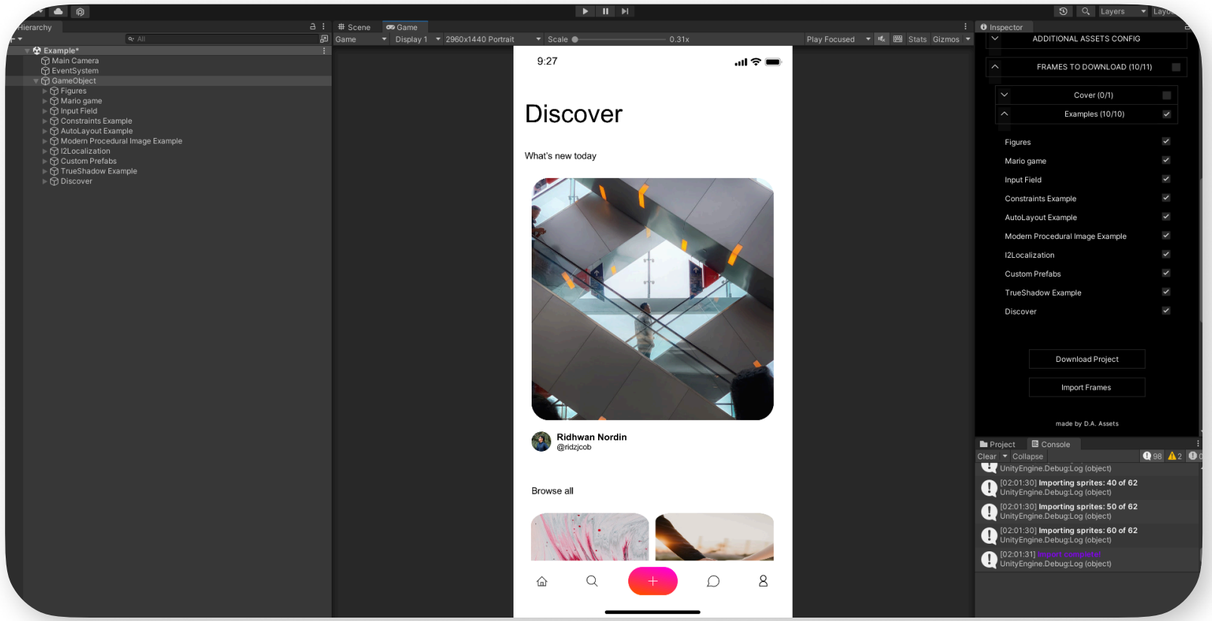
At the moment, the asset does not support importing **Sections** due to API limitations. To import the contents of **Sections**, place them in a **Frame**.

Import Frames

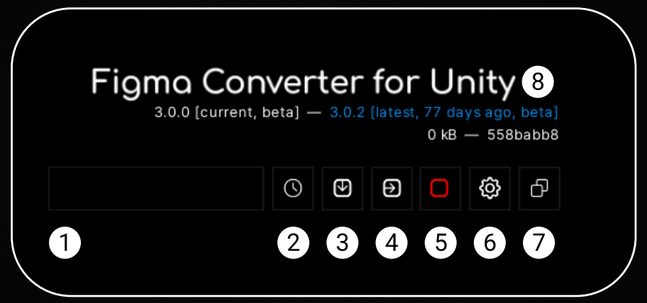
5 Press on the "**Import**" button, to start the import.



6 At the end of the import, you will see a message in the console - "**Import complete!**".



ASSET UI



1. Link to your project in figma.

2. Open the list of cached projects that you have previously imported.
   From the list, **you can select** the **project** you want to import.

3. Download the project from the link.
   All downloaded projects are automatically cached.

4. Import selected frames from the downloaded project.

5. Stop import.

6. Open asset settings.

7. Switch the asset display mode. Available modes:
   - In the inspector
   - Windowed

8. Label displaying the current and latest version of the asset.

   If the version is colored **blue**, it means that too much time has passed since the release of the latest version, and you are **recommended to update** the asset.

   If the version is colored **red**, it indicates that it contains errors, and it is strongly **recommended to update** the asset.

   If you hover your cursor over the version, you will see a tooltip with detailed information about that version.

MAIN SETTINGS

1. **UGUI** - layout import into **Canvas**.
   UITK - layout import into **UI Builder**.
   NOVA - layout import into **Nova UI**.

2. Sets the Layer value for all imported GameObjects.

3. **ABSOLUTE** - positioning of frames on the canvas as in Figma.
   GAMEVIEW - anchoring frames to the edges of GameView (does not work in UITK mode).

4. 

   The value that will be assigned to all imported GameObjects.

5. Maximum length of GameObject names.

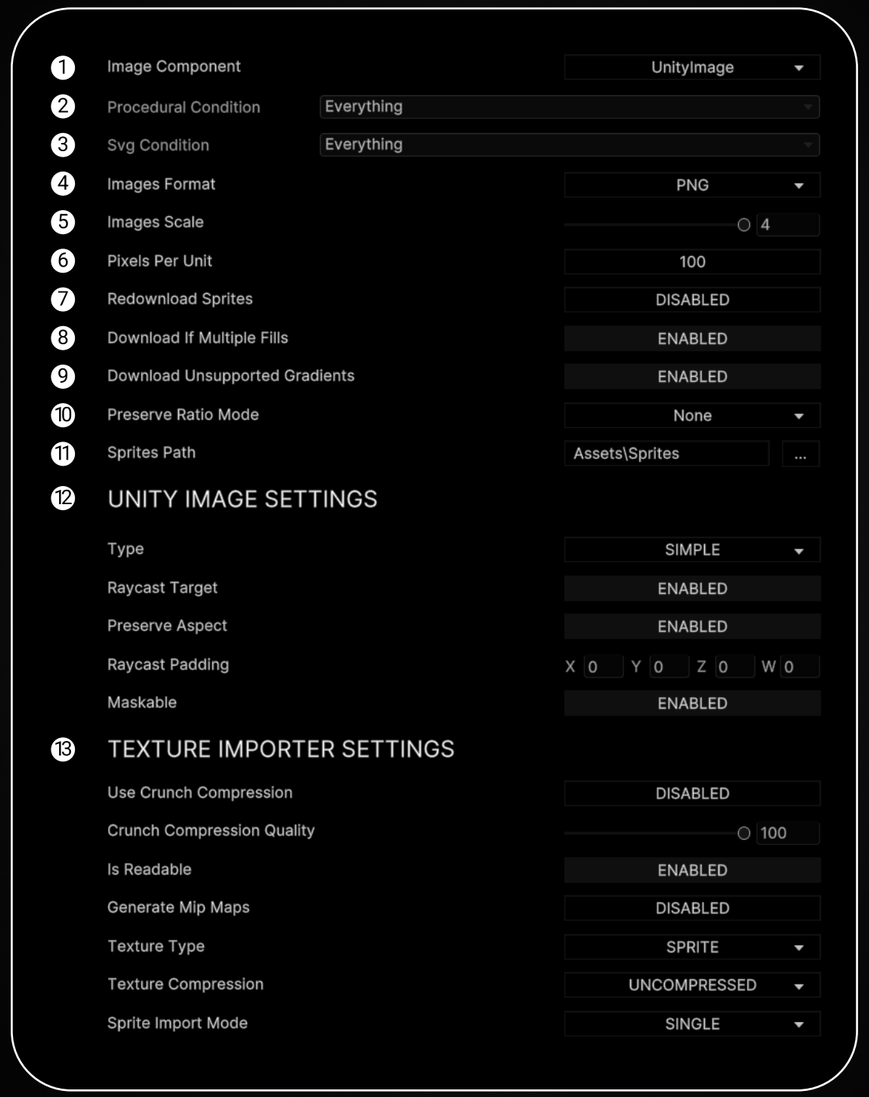
6. Maximum length of names for GameObjects containing a text component.

7. If enabled, your project is imported "as is", i.e., without "smart" merging of individual vectors into single sprites.
   The function is in beta stage, and errors may occur during its using.

8. Enable or disable HTTPS when accessing Figma servers in case of certificate issues.

8. Characters, aside from Latin letters and numbers, that may appear in GameObject names.
   Some characters will be ignored in certain cases, such as when a backslash is used in a sprite name.
   If you add new characters to this list, the stable operation of the asset cannot be guaranteed.

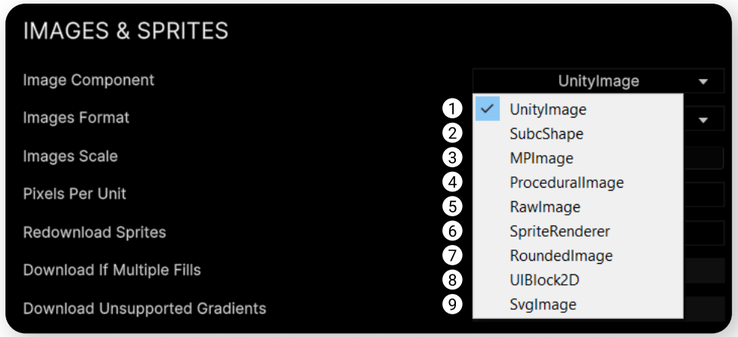
IMAGES & SPRITES



1 **Image Component** — The component used to render sprites in your scene.

You can view the current list of supported image assets in the "DEPENDENCIES" tab on the asset's page.

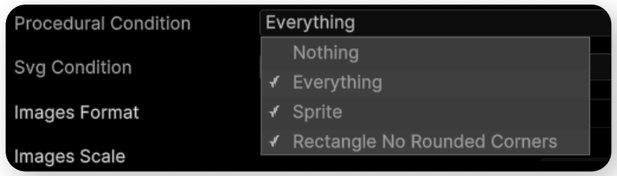
For design-specific details about working with these assets,
you can read the "**Manual for designers**":
Assets\D.A. Assets\Figma Converter for Unity\Manual for designers.pdf



1. UnityEngine.UI.Image - Built-in;
2. Shapes2D.Shape - from Shapes2D asset;
3. MPUIKIT.MPImage - from Modern Procedural UI Kit asset;
4. UnityEngine.UI.ProceduralImage - from Procedural UI Image asset;
5. UnityEngine.UI.RawImage - Built-in;
6. UnityEngine.SpriteRenderer - Built-in;
7. DTT.UI.ProceduralUI.RoundedImage - from Procedural UI asset;
8. Nova.UIBlock2D - from Nova asset;
9. Unity.VectorGraphics.SVGImage - from Vector Graphics asset.

Images
Sprites

2 **Procedural Conditions** — These are cases where the UI.Image component will be used instead of a procedural image component:
a) If your component was imported as a sprite, because it is a sprite rather than a simple shape that can be drawn procedurally.



b) If it is a simple rectangle without rounded edges, which can also be drawn by UI.Image.

3 **Svg Conditions** — These are cases where the UI.Image component will be used instead of a vector image component:

a) If it is an Image, Emoji, or Video.
b) If your component or its children contain any effects.

According to the Vector Graphics asset documentation, effects in vector images are currently not supported.

4 **Image Format** — The format of the downloaded images. Can be **PNG** or **JPG**.

5 **Image Scale** — The scale of the downloaded images.
This option is identical to the same option when exporting image from Figma.

6 **Pixels Per Unit** — The value that will be assigned to all imported sprites. When using the “**SpriteRenderer”** component, this value is ignored, and instead, the “**Image Scale**“ value is used for sprites.

7 **Redownload Sprites** — If enabled, the asset downloads and overwrites sprites with each import, even if they have already been downloaded.

Images
Sprites

8 **Download if Multiple Fills** — If enabled, if your component has multiple fills or fill + outline simultaneously, it will be downloaded.
If disabled, the asset will attempt to render this Figma component using the selected Unity component (e.g., UI.Image).
In other words, the Figma component will not be downloaded.

9 **Download Unsupported Gradients** — If enabled, if the selected Unity component does not support a certain type of gradient, the Figma component will be downloaded as a PNG/JPG image.
If disabled, the asset will attempt to render this Figma component.

10 **Preserve Ratio Mode** — If enabled, all **Image** components will have the "**Preserve Ratio Mode**" feature enabled during import.

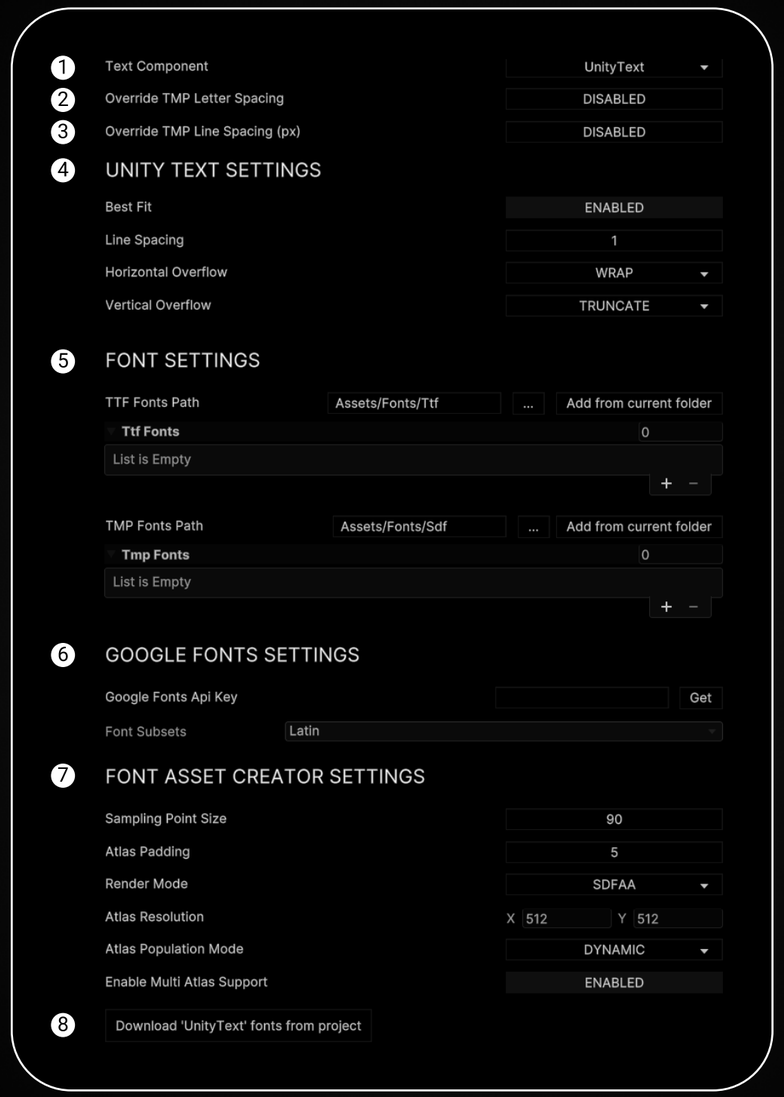
11 **Sprites Path** — The **path** to the folder where the **sprites** are downloaded.
You can **set your own** path by clicking the button with **three dots**.

12 Special settings for the component selected in "Image Component."
This section will change depending on the chosen component.

13 **Texture Importer Settings** — Settings applied to your sprites in the Sprite Inspector.
You can read more about TextureImporter settings in the TextureImporter documentation.

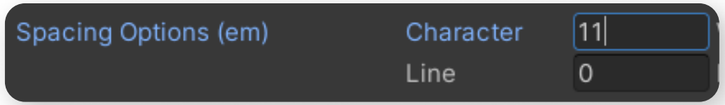
If you are using the "SVG Image" component, the "SVG IMPORTER SETTINGS" section will be visible.
You can read about the SVG Importer settings in the Vector Graphics documentation.

TEXT & FONTS



1. **Text Component** — The component that will be drawn on the scene when importing **texts** from Figma.
   Available components:
   - **UNITY TEXT** (built-in)
   - **TEXTMESHPRO** (need **TextMeshPro** asset)
   - **RTLTMPro** (link)

2. **Override TMP Letter Spacing** — By default, the "**Letter Spacing**" value is not imported from **Figma** to **Unity**, as it is recommended to set this value manually in the **TextMeshPro** font file. If you enable this feature, the "**Letter Spacing**" value will be transferred from Figma to Unity for each individual text component.



3. **Override TMP Line Spacing** — The same as in point 2, but for "**Line Spacing**".

4. **Unity Text Settings** — Special settings for the text component selected in "**Text Component**". This section will change depending on the chosen text component.
   You can read more about these settings in the "**Text**" or "**TextMeshPro**" documentation.

5. **Font Settings** — In this section, you can add fonts that the asset will use during the import of your project.
   The asset will pull fonts from serialized arrays, so you need to fill one of them based on your text component.
   You can place fonts in the folder specified in the field and click the "Add Fonts from Current Folder" button to automatically load fonts from this folder into the serialized array.
   You can specify a custom font folder using the button with the ellipsis.
   The font folder must be located inside the "Assets" folder.

Text
Fonts

6 **Google Fonts Settings** — By connecting Google Fonts to Figma Converter for Unity, you can automatically download fonts during the import of your Figma project.
This will only work for fonts available in the Google Fonts repository.
You can learn more about this feature on the "Google Fonts" slide.

7 **Font Asset Creator Settings** — When importing fonts from Google Fonts, if you have TextMeshPro enabled, these settings will be used during the automatic conversion of regular fonts to TextMeshPro fonts.
You can learn more about these settings in the Font Asset Creator documentation.

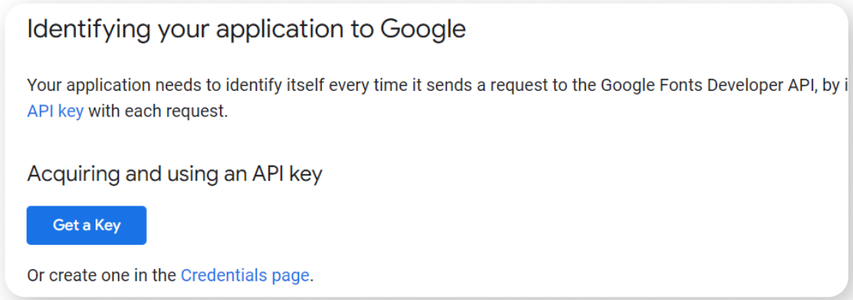
8 Button for **downloading** and generating **missing fonts without importing** frames.
To use the button, you **need** to **select** a text **component** in the “**UNITY COMPONENTS**” tab, and then download the project using the download button (the import button does not need to be pressed).

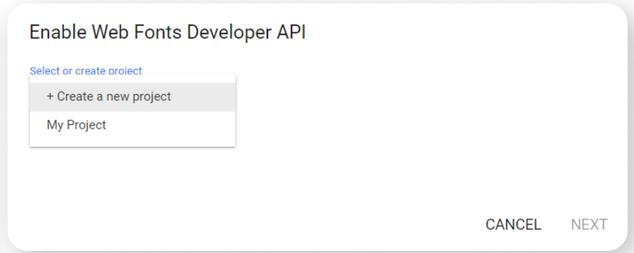
Google Fonts

Some fonts might be missing from the Google Fonts repository.
In that case, they won't be downloaded automatically, and you will see an error in the console.
You'll need to manually import those fonts into your project.

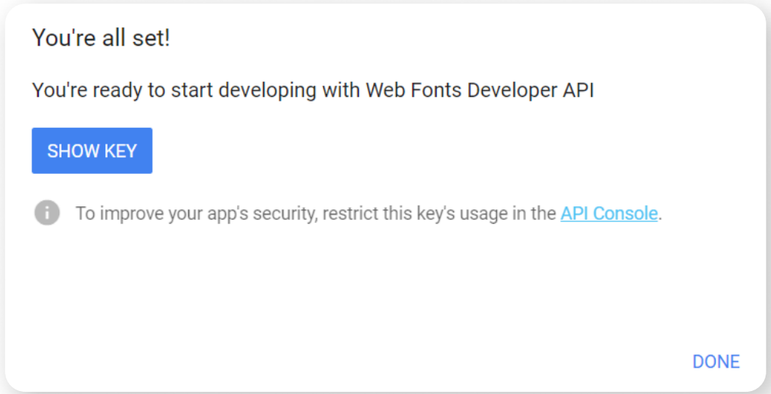
1. In order for the asset to automatically download missing fonts, you need to obtain a Google Fonts API key.
Go to: https://developers.google.com/fonts/docs/developer_api#identifying_your_application_to_google

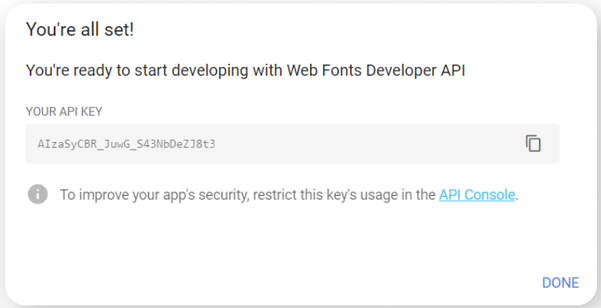
2. Click on “**Get a Key**” button, then create new project or select existing.





3. Click on “**SHOW KEY**” button, then copy your api key.





4. Open the "**TEXT & FONTS**" tab and paste the obtained key into the "**Google Fonts Api Key**".
Now your fonts will be automatically downloaded from the Google Fonts repository.

LOCALIZATION

1. **Localization Component** — The component that will be used to localize your text components.
   Available components:
   - **D.A. Localizator** (soon)
   - **I2 Localization** (link)

   Instructions for setting up both assets to work with **Figma Converter for Unity** are available in this manual under the relevant sections.

2. **Localizator** (only for D.A. Localizator) — The component that will be used to localize your text components.

3. **Localization Key Max Length** — The max length of the localization key.

4. **Localization Key Case** — The localization key case. Available options:
   - snake_case
   - UPPER_SNAKE_CASE
   - PascalCase

5. **Layout Language** — The language of your Figma layout. If this is "en-US," localization key values will be entered in the "en-US" column of your CSV table.

6. **Separator** — The separator of your CSV file.

7. **Localization Folder Path** — The folder where your localization file is located. It must be inside the "Resources" folder before import. You can change its location after import.

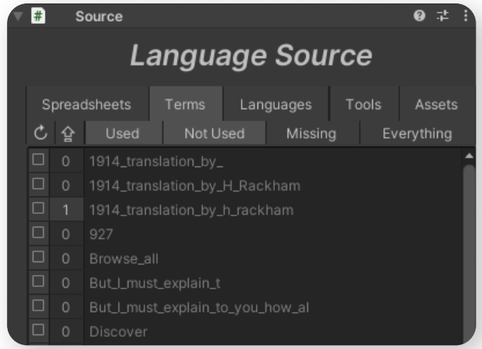
8. **Localization File Name** — The name of the localization file that will be in the folder.

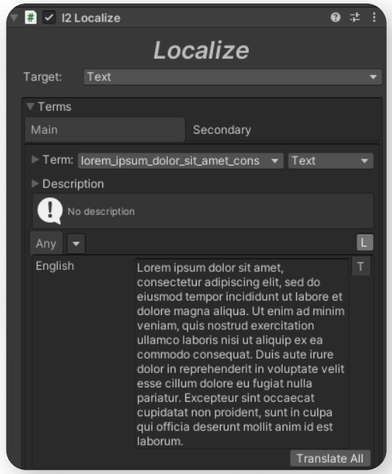
I2Localization

You can purchase "**I2Localization**" asset and use it in conjunction with "**Figma Converter for Unity**".

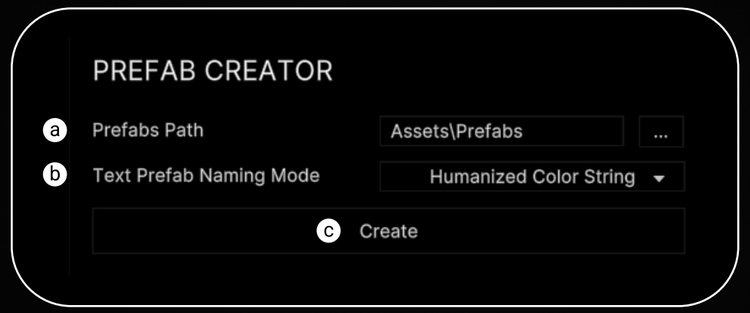
Video manual:
https://www.youtube.com/watch?v=Rn_Fv-oory8

1. Add the purchased asset to your project.
2. When importing assets, follow the instructions in the "**Dependency Manager Manual**".
3. After a successful asset import, in the "**LOCALIZATION**" tab, switch the "**Localization Component**" to "**I2 Localization**".
4. Import your layout as you normally would.
5. **After import**, script "**I2Localize**" will be **added to all text components**, their text will be written to the localization file "**Localization.csv**", the localization corresponding to the text component will be **selected** in the script.
   All further **instructions** are detailed in the **manual** for "**I2Localization**" asset.





CREATING PREFABS



a) Folder where prefabs will be saved when creating prefabs using the asset.
You can set your own folder by clicking the button with three dots.

b) Naming type for text prefabs. Modes:

- Humanized Color String - The name of the prefab includes the name of the most suitable color, which is determined automatically.
  Example of a name: "TextMeshPro white 12px".
- Humanized Color HEX - The color is indicated in HEX format in the prefab name.
  Example of a name: "TextMeshPro #0C8CE9 12px".
- Figma - The text is named the same as its component in Figma.

c) Button to create prefabs.

You can create prefabs for your imported objects under the following conditions:

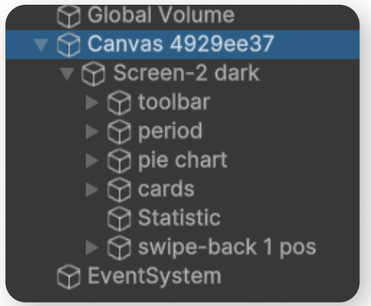
1. All your imported **components have** a **SyncHelper** component.
2. You have **not duplicated/copied** game objects that contain the **SyncHelper** component.

You have **two options** for creating prefabs:

1. You **imported** the **entire frame** and will create prefabs for the whole frame in automatic mode.
2. You **imported** your **Master Components** from Figma, created prefabs for them, and only then imported your frames. In this case, during import, the asset will use the prefabs created by the asset to draw **specific** layout **elements** on the Canvas, while other elements will not be prefabs. This option is suitable if you want to create prefabs **only for certain** elements.

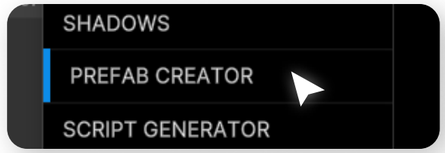
Before each attempt to create prefabs, the asset will **automatically** create a **backup** of your scene. For more information about backups, refer to the relevant section of this manual.

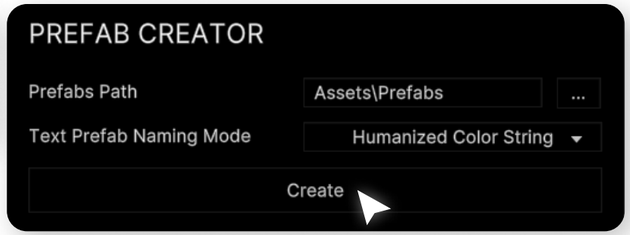
1. To start creating prefabs, go to the Figma Converter settings located on the canvas whose child objects you want to convert into prefabs.



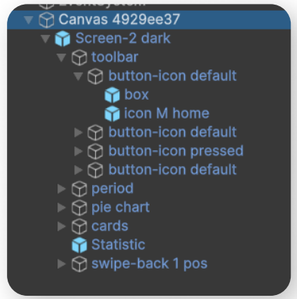
Creating prefabs

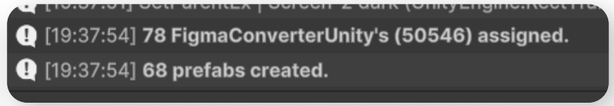
3 Go to the "**PREFAB CREATOR**" tab.
Configure the prefab creation options and click the "**Create Prefabs**" button.





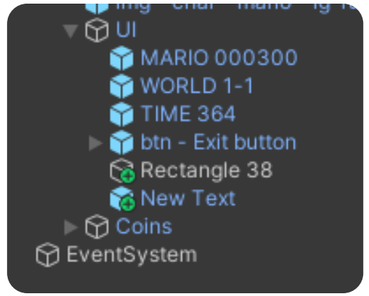
4 After the algorithm runs for a while, you'll see a console message indicating that the prefabs were **created successfully**.





5 Keep in mind that if, after creating prefabs, you change the Image Component, Text Component, or Button Component and then perform an import using the existing prefabs, this will lead to **errors** during the layout import.

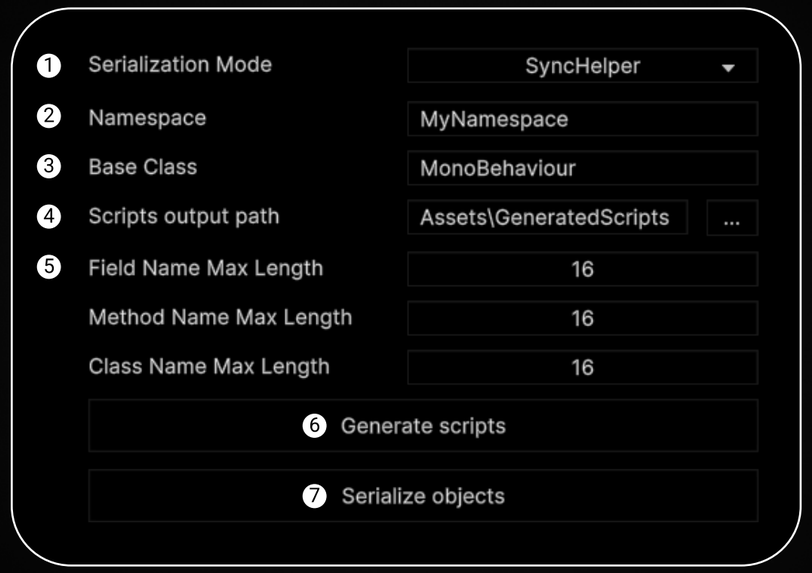
3



If there are already objects on your scene that are prefabs, but changes have been made to those same objects in Figma,
and you want to update them on your scene - **you can** do so.

New objects will be added to your existing prefabs during the next import, but locally saved prefabs in the assets will not be updated.

SCRIPT GENERATOR



After import, you can **generate scripts** for your frames and/or **automatically serialize** game objects within them.

Generating scripts from prefabs created with the Figma Converter is
currently not supported - you need to generate scripts before creating the prefabs.

1. **Serialization Mode** –The principle by which GameObjects will be serialized into script fields. Two options are available:
   - **SyncHelpers** – Since scripts are generated based on SyncHelpers, this serialization method assigns all GameObjects to exactly those serialized fields for which they were generated in the code.
   - **GameObjectNames** – Game objects will be serialized into fields whose **names match** the game object **names**.
   - **Attributes** – Game objects will be serialized into fields that have the attribute "**FObjectAttribute**", with text that matches the name of the component in Figma for which the game object was created.

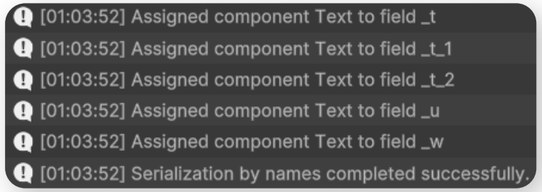
   Below in the manual, you can find instructions for all serialization options or script generation.
2. Namespace for generated scripts.
3. Base class for generated scripts.
4. Folder where generated scripts will be saved.
5. 4-6 is the maximum length for field, method, or class names.
6. Button for generating scripts.
7. Button for serializing GameObjects into scripts.

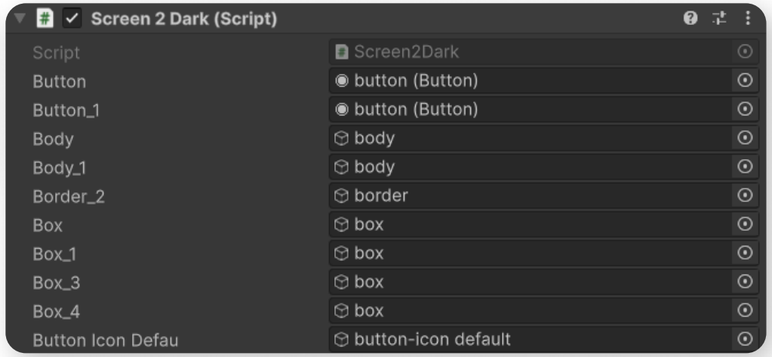
Serialization by SyncHelpers

1. To generate scripts and serialize them by SyncHelpers, import your project and click the "Generate Scripts" button.

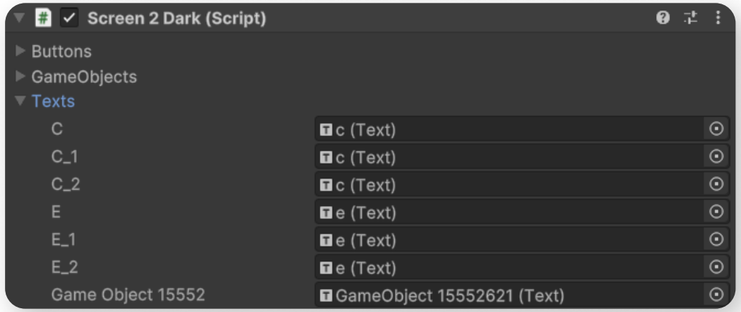
2. Once your scripts are generated and the project recompiles, switch the "Serialization Mode" field in the script generator settings to "SyncHelper" and click the "Serialize Objects" button.

3. After this, the asset will automatically add the generated scripts to the corresponding frames and serialize the objects into them, as shown in the screenshot.





3. **Additional point.** As we can see, there are quite a lot of serialized fields (the screenshot is cropped, but there are dozens of them). You can use a custom inspector that groups serialized objects. It looks like this:



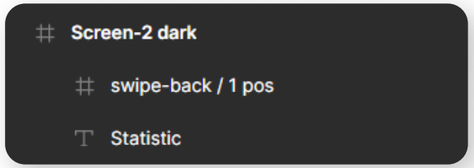
To apply this sorting to any of your scripts, you need to use the script from this gist.
You should also learn how to create a Custom Inspector for your own script.

D.A. Assets

Serialization by Attributes

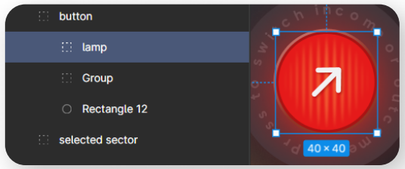
1. If you have your own scripts, you can skip the generation step.

2. Add the attribute [FObject("**frame_name**")] to your class, where **frame_name** is the name of the frame in Figma to which this script belongs.




```
[FObject("Screen-2 dark")]
Unity Script | 0 references
public class MyCustomClass : MonoBehaviour
```

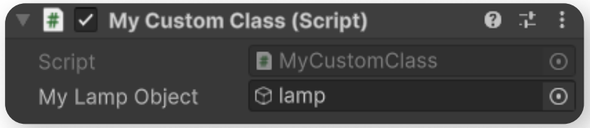
3. Add the attribute [FObject("**component_name**")] to your serialized fields, where **component_name** is the name of the component in Figma, whose game object should be serialized into this field.




```
[FObject("Screen-2 dark")]
Unity Script | 0 references
public class MyCustomClass : MonoBehaviour
{
    [SerializeField, FObject("lamp")] GameObject myLampObject;
}
```

3. Click the "Serialize Objects" button.

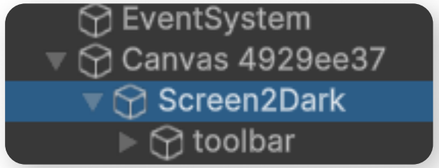
4. Game objects will be serialized into your fields according to your settings.

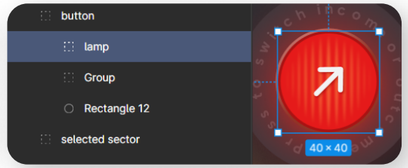


Serialization by GameObject name

1. If you have your own scripts, you can skip the generation step.

2. The name of the GameObject of the frame must match the name of the script.
   The name of the GameObject to be serialized into the field must match the name of the field.

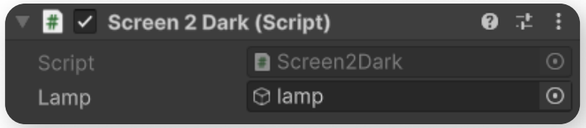





```
Unity Script (1 asset reference) | 0 references
public class Screen2Dark : MonoBehaviour
{
    [SerializeField] GameObject lamp;
}
```

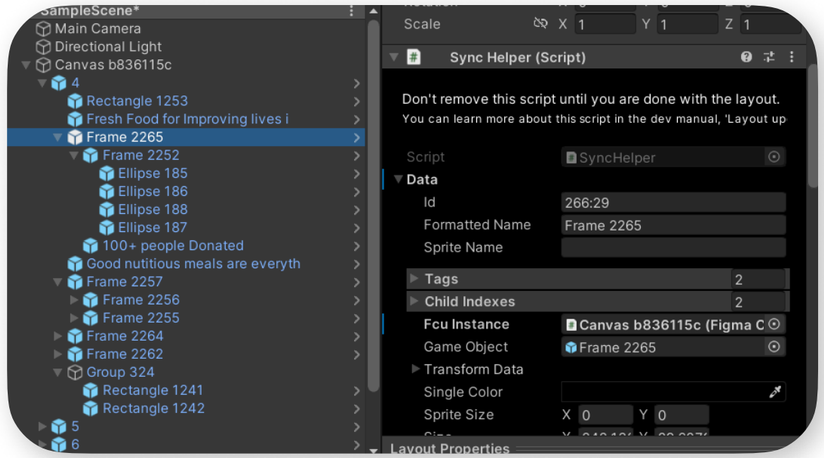
3. Click the "Serialize Objects" button.

4. The asset will serialize GameObjects into fields that match the GameObject's name.



Layout updating

1. After importing the project, a script "**SyncHelper.cs**" will be added to all imported objects.
This script is needed to synchronize objects between Figma and Unity during the import.



2. You can perform a re-import of your project to update the transforms and properties of objects (color, text), as well as add new objects and remove missing ones.

3. If you've created prefabs from your imported objects, during subsequent imports, you'll be able to update only their transforms and properties. New objects within the prefabs won't be added.

4. Avoid creating duplicates of objects with the same "**SyncHelper.cs**" script in your Unity scene.
To duplicate an object, create its duplicate in Figma, and then repeat the import process.

5. Do not remove the "**SyncHelper.cs**" scripts until you have finished importing your project from Figma to Unity.

6. After you've completed your work on the project and are sure you won't be updating it further, you can remove the "**SyncHelper.cs**" scripts from your objects using the corresponding function in the asset's context menu (more details in the "**Context Menu**" section).

LAYOUT UPDATING

If **previously imported components** are present on your scene, attempting a new import will open the **PreImportWindow**.
With its help, you can more precisely adjust your new import, specifically - **update** the **existing** components on the scene (**synchronize** them **with Figma**), see how the imported components differ from those in Figma and **compare** their **properties**, as well as **remove** unwanted components **from the scene**.

Below you will find a description of the PreImportWindow interface elements with an explanation of its functionality.

1. The section where you can analyze and configure the import of components to the scene.
2. Button to open the "**Diffchecker**" website.
3. Components that exist in the Figma project but not on the Unity scene (new components).
4. Components that have been changed in Figma since the last import.
5. Components that have been changed in Unity since the last import.

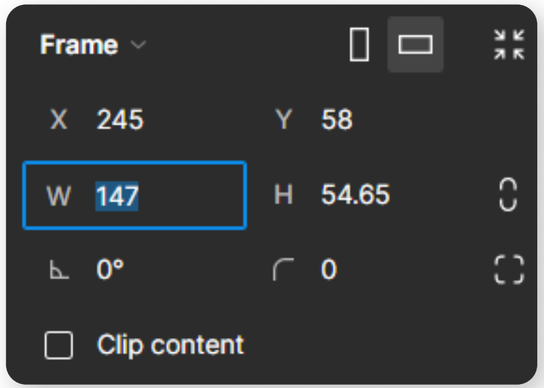
Layout updating

6 Components that have not changed.

7 The RectTransform size in Unity differs from the component size in Figma.

8,9 Lists of all properties of a Figma component.
Old data - data of the component currently on the scene.
New data - data of the component that is only in Figma and has not yet been imported onto the scene.

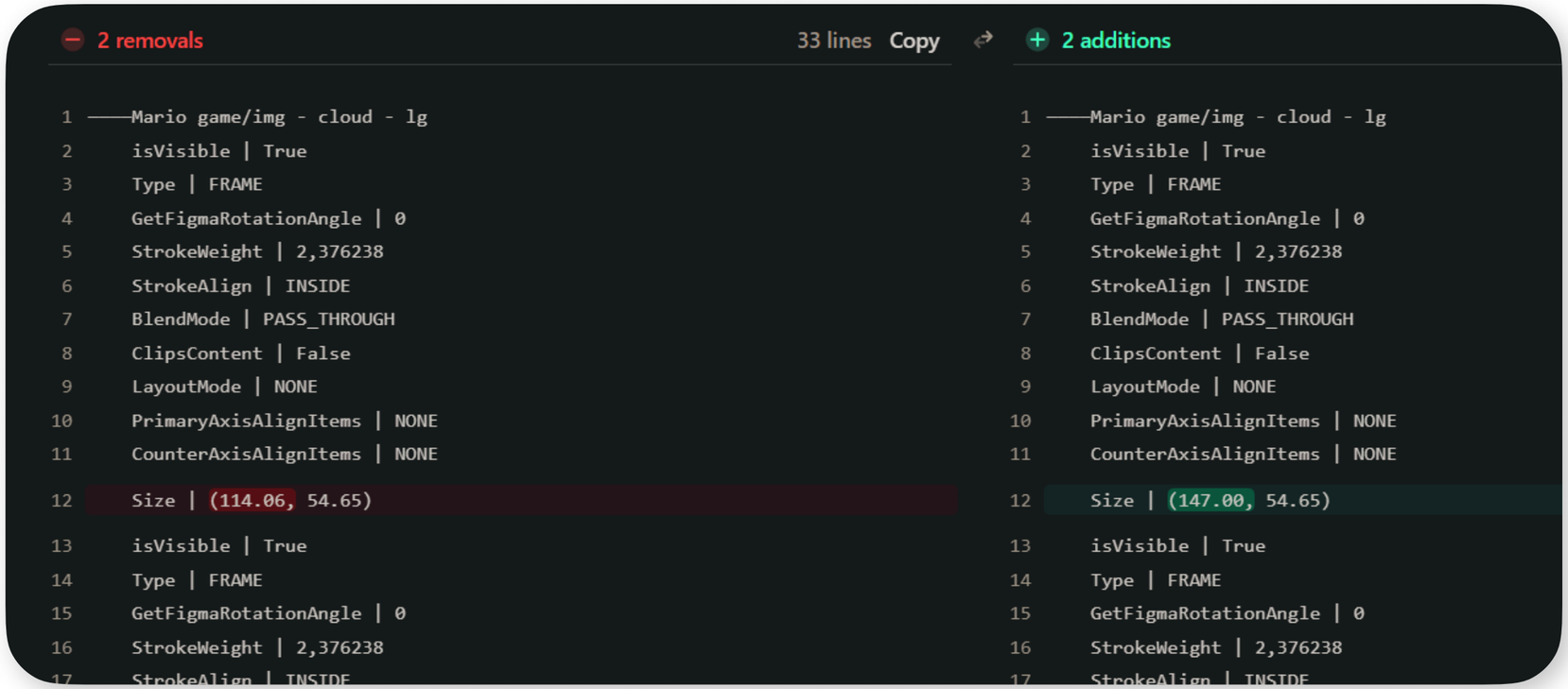
The **data about the properties** of the Figma component is **captured at the time of import** and remains unaffected by any changes to the GameObject on the scene.
In some cases, you may need this information, for example, made manipulations with the component in Figma that was imported into Unity as a sprite, and you need to decide whether to update the sprite during re-import, or not.



If this data differs, you will see the text "**Component data has differences**" in the component item.
To see how the new component differs from the existing one, you can open the website "**Diffchecker**" using the "**Open difference checker website**" button, and sequentially copy the **Old** and **New** data into the "**Original text**" and "**Changed text**" fields, then press the "**Find difference**" button.

In the example below, I changed the size of the component in Figma but did not change the RectTransform in Unity.

Layout updating



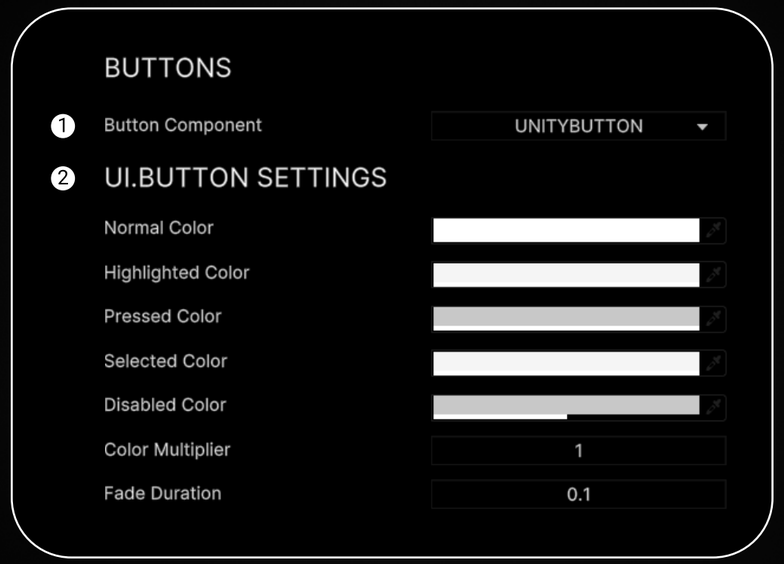
10. A message about data differences, related to points **7** and **8**.

11. The **Color** property of **Graphic** component in Unity differs from the component color in Figma.

12. In this section, you can select frames or individual components that will be removed from the scene during the new import.

13. Click this button to continue the import with the parameters you have selected.
By default, if you have not made any changes in **PreImportWindow** - all components that are both on the scene and in Figma are synchronized, new components are imported, and components from the old import are not deleted.

BUTTONS



1. **Button Component** — The component that will be drawn on the scene when importing objects with "**btn**" tag from Figma.
   Available components:
   - **Unity Button** - built-in component.
   - **D.A. Button** - supports **multiple TargetGraphics**, **color**/**size**/**position** animations for various states via AnimationCurve. Supports **sprite swapping** and **looped** animations.
     Information on using **D.A. Button** can be found in the manual attached to **D.A. Button** asset.
     Sold separately.

2. **UI.BUTTON SETTINGS** — Special settings for the text component selected in "**Button Component**".
   This section will change depending on the chosen button component.
   You can read more about these settings in the "**UI.Button**" documentation.

3. Information on setting up button component in Figma can be found in the **Manual for Designers** in the relevant section.

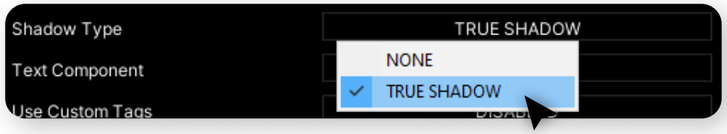
SHADOWS

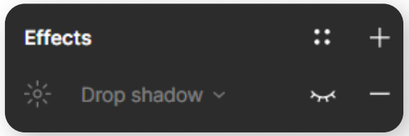
1. The asset supports two methods of importing shadow components:
   - **FIGMA**: The shadow is part of the downloaded sprite.
   - **TRUESHADOW**: The shadow is **procedurally rendered** using the asset.

2. If you want the component's shadow not to be part of your sprite but rendered procedurally, you can use the **TrueShadow** asset.

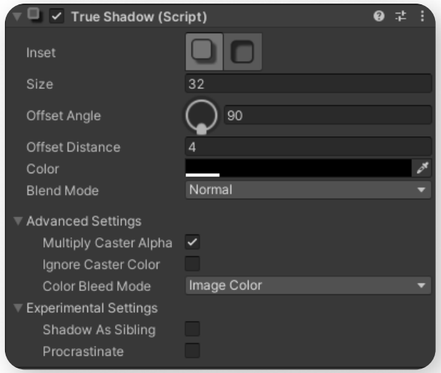
3. When importing asset, follow the instructions in the "**Dependency Manager Manual**".

4. In the "**SHADOWS**" tab switch parameter "**Shadow Type**" to **“True Shadow**".



5. 

Before importing your layout using the **“True Shadow”** mode, you need to make all the shadows in your Figma project invisible.

6. 

After import, **all** your **components** that have a **shadow** in the Figma layout **will have** a **shadow** script from "**TrueShadow**" asset.

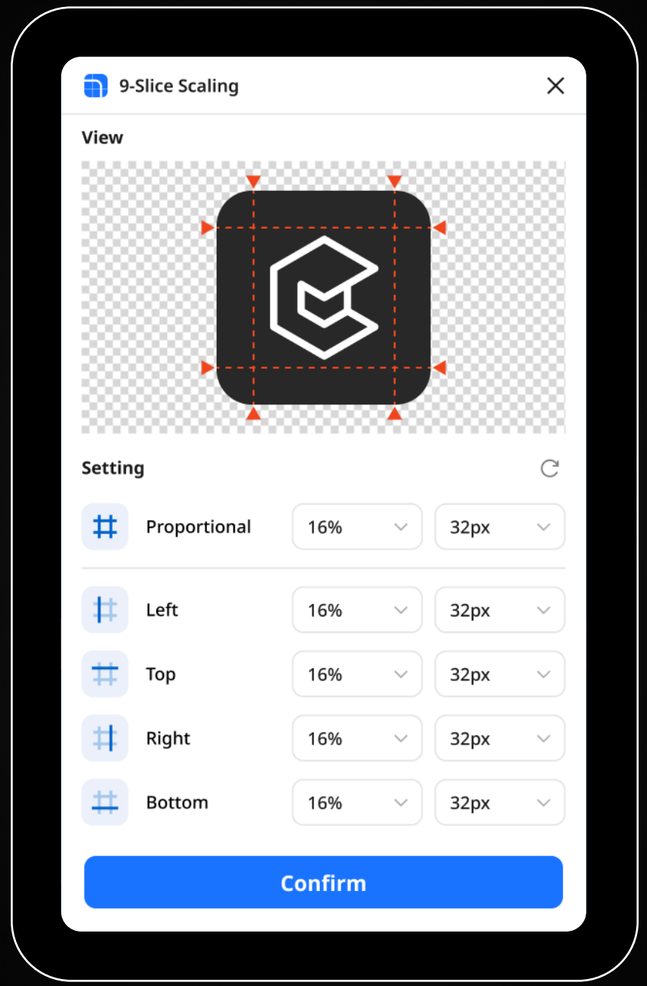
To use this functionality properly, **read** the section of this asset in the "**Manual for designers**".

UI TOOLKIT

1. **UITK Linking Mode** — The method by which the search and linking of components in **UI Toolkit** will be performed.
   You can learn more about this in the manual included with the "**UITK Element Linker**" asset.

2. **UITK Output Path** — The folder where the result of the import into **UITK** will be saved.

SPRITE SLICE

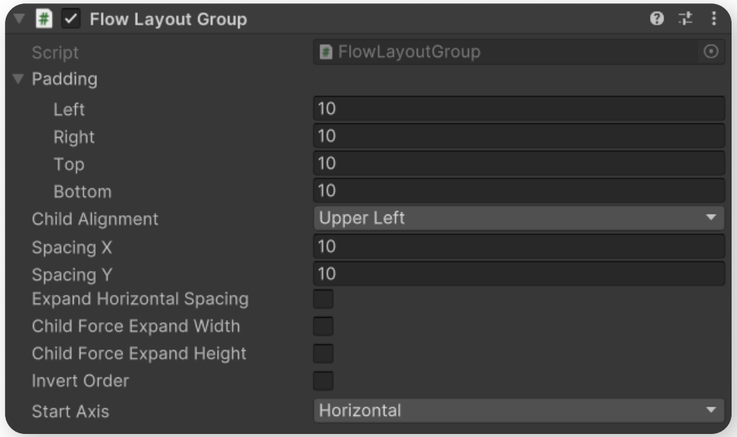


1. The asset supports automatic transfer of slices from Figma to Unity.
Your designer should use the "**9-Slice Scaling**" plugin to create the necessary slices in Figma according to the plugin's instructions.
On the developer's side, there is no need to enable or change anything; the values will be transferred automatically during import.

GridLayoutGroup

1. Since the standard "**UI.GridLayoutGroup**" component does not replicate the behavior of the Figma GridLayoutGroup, the asset uses the "**FlowLayoutGroup**" component from the "**Unity UI Extensions**" asset instead.

   To enable the asset to use "**FlowLayoutGroup**" when importing your layout, download and install the "**Unity UI Extensions**" asset, and then activate the "**Unity UI Extensions**" dependency in the "**Dependency Manager**".



Nova UI

Currently, the following are not supported when importing using Nova UI:

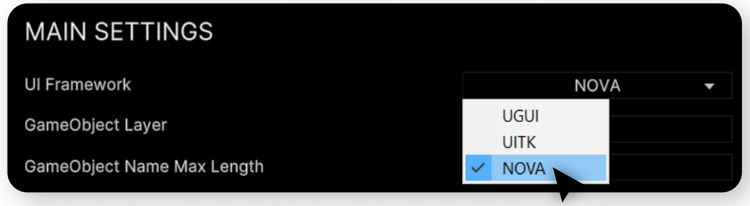
1. Prefab creation using FCU;
2. Anchors (constraints);
3. Auto layouts;
4. Layout updating using FCU.

The lack of support for the listed functions will not affect the appearance of your layout.

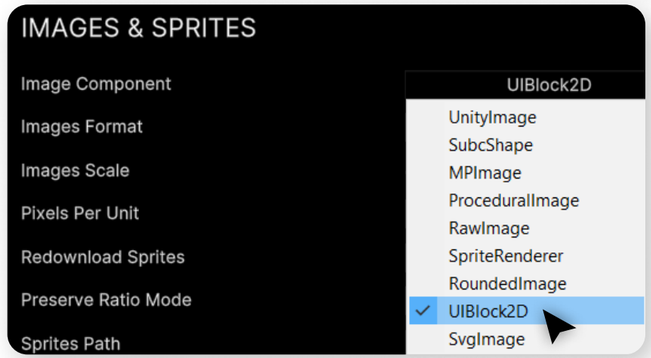
However, after the import, if necessary, you can configure all these things manually.

1. You can use the "**Nova UI**" framework instead of the standard "**UGUI Canvas**" or "**UI Toolkit**".

2. To import your "**Figma**" layout using "**Nova UI**" components, import "**Nova UI**" and "**TextMeshPro**" into your project.

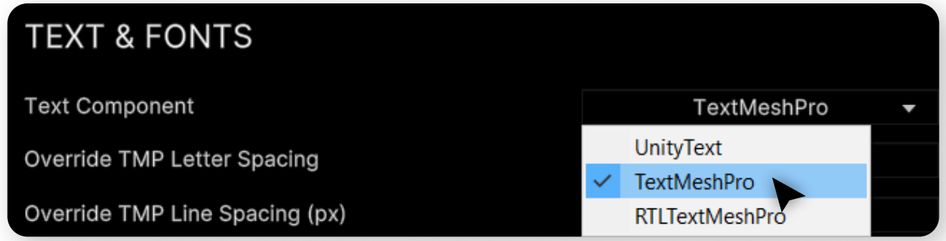
3. 

Switch the "**UI Framework**" in the "**Main Settings**" of the asset to "**NOVA**".

3. 

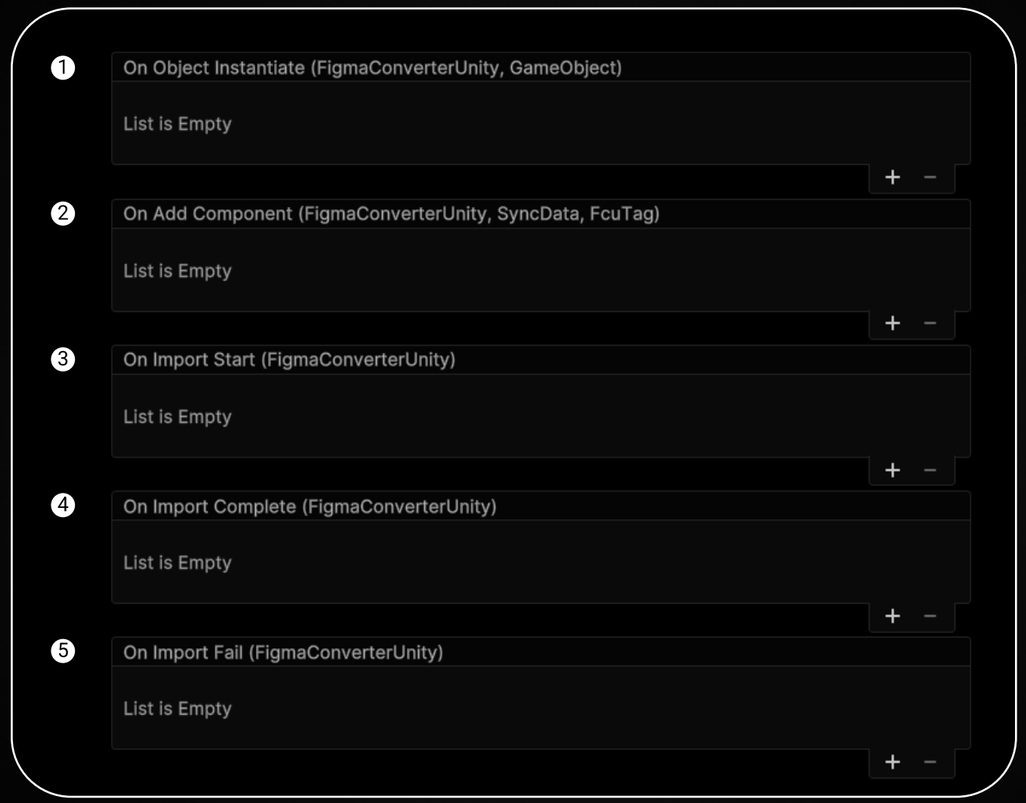
Switch the "**Image Component**" in the "**IMAGES & SPRITES**" tab to "**UIBlock2D**".

4. In the asset settings, under the "**NOVA COMPONENTS**" tab, switch the "**Text Component**" to "**TextMeshPro**".
The integration with "**Nova UI**" will only work if you use "**TextMeshPro**" as the text component during the import.



5. After this, you can import your layout following the general import instructions.

IMPORT EVENTS



Import Events can be used to customize the import process.
For example, if you want to add your own components to GameObjects according to specific algorithms during import.

1. **On Object Instantiate** — Called when creating a GameObject in the scene.

   How can you use this event? For example, you can parse the GameObject's name, which is returned in this event, and based on that, perform certain actions—such as adding your own custom script to this GameObject.

2. **On Add Component** — Called when adding a specific component to a GameObject during import.

   In this case, FcuTag is a special tag that the Converter assigns to each imported object.
   As you know from the Tags section of the Manual for Designers, tags can be manually set in the names of your objects in the Figma layout.

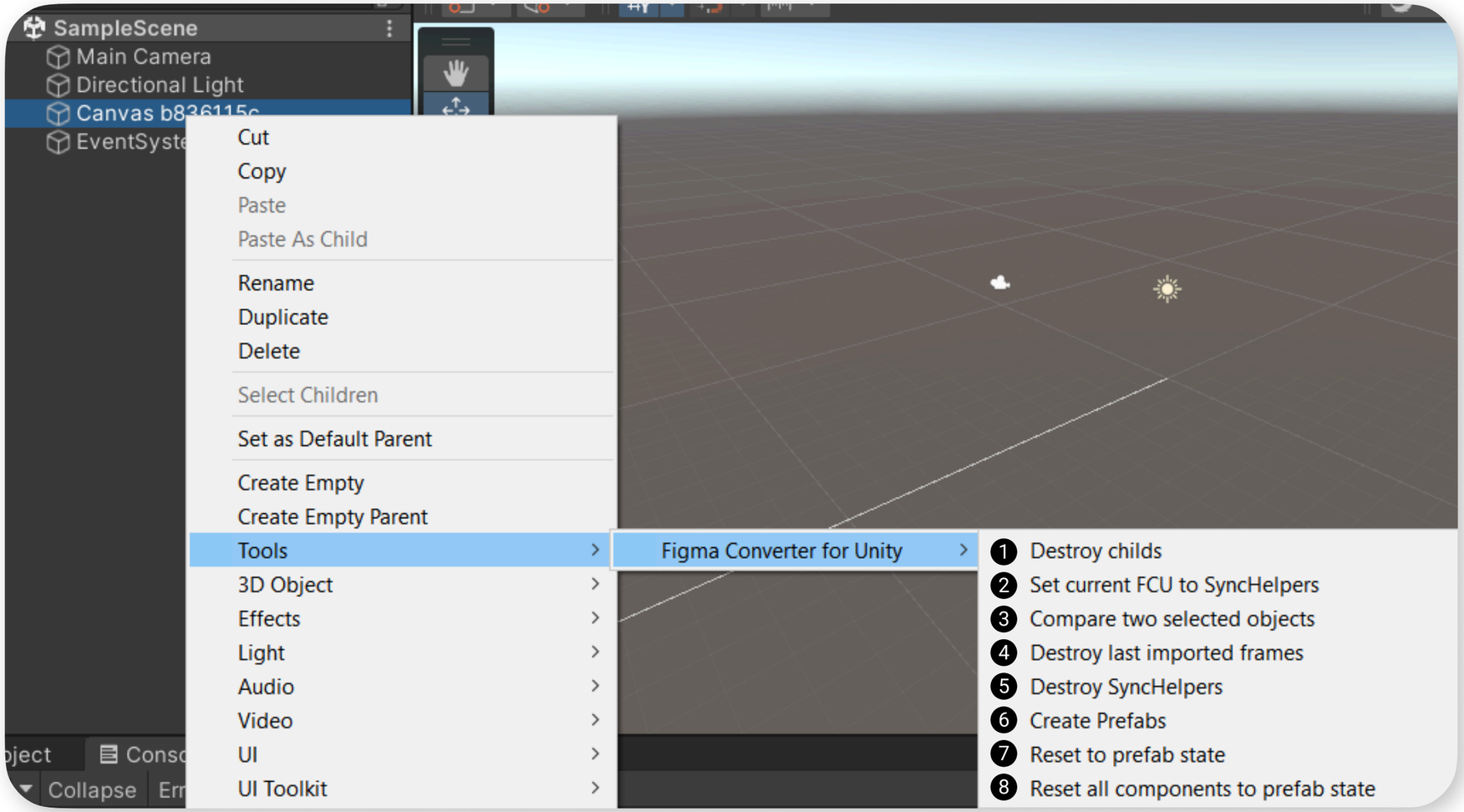
   So, if FcuTag has a value like "FcuTag.Text," it means that the current component triggering this event is a text component.

3. **On Import Start** — Called before the start of import.

4. **On Import Complete** — Called after the import is complete.

5. **On Import Fail** — Called if the import is stopped.

Context menu



1. Deletes the child objects of the current canvas.

2. Assigns the main script of the asset to the serialized field of child objects of the current canvas. This is necessary for creating prefabs and updating the project during re-import.
Works only for objects that have the SyncHelper.cs script attached to them.

Context menu

3. Compares two objects that have the SyncHelper script attached to them. Using this function, you can determine the differences between two objects to avoid duplication in your Figma and Unity projects.

4. Removes the last imported frames. Please note that this function is temporarily not operational.

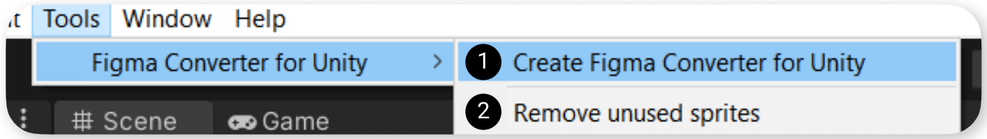
5. Removes the SyncHelper.cs script from all child objects of the current canvas. Please note that after removing these scripts from objects, you won't be able to synchronize your Unity project with the Figma project. Only delete SyncHelper.cs if you are certain that you won't need to synchronize your layout anymore.

6. Creates prefabs from the objects of the current canvas. Creating prefabs is only possible if all objects on the canvas have the SyncHelper.cs script attached.

7. Resets the selected GameObject to the state of the prefab. Child objects are not reset.
Resets the selected object and all its child objects to the state of the prefab. The SyncHelper.cs script is not needed for these functions to work.

8. Resets the selected GameObject and all its child GameObjects to the state of the prefab. The SyncHelper.cs script is not needed for this function to work.

Context menu

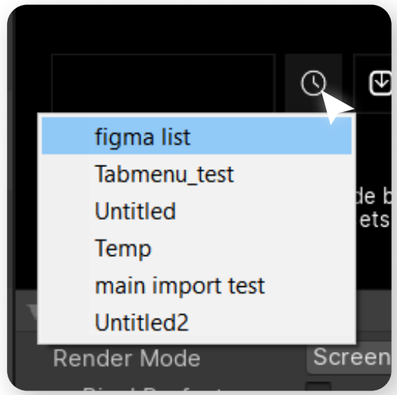


1. Creates a GameObject with the FigmaConverterUnity script on the scene.

2. Opens a window where you can specify a folder containing your sprites, from which you want to remove all sprites that are not used in the Image components of all objects in the current open scene.

Scene backups and project cache

1. The folder with backups of your active scene is located here: **Library\Backup\Scene**
2. Backups are automatically creates before each import and before creating prefabs.
3. A backup is created for a **previously saved local scene file**. If you see a asterisk (*) next to the project name in the Unity interface, it indicates that changes you made to the scene without saving it will not be included in the backup.
4. If you have never saved your current scene (the file of your scene is not present on the disk),
the scene will be automatically saved before importing at the path "**Assets/Scenes/time_scene_name.unity**".
5. With each project download, the transform and properties of objects from your Figma project are cached.
To avoid downloading it again, you can choose the cached version from the dropdown menu.



Import Issues

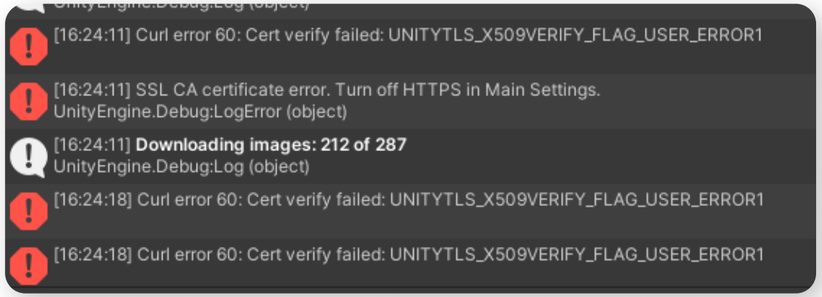
This section will be updated.

1. My frame doesn't look the same after import as it does in Figma's layout. Why?
The answer to this question can be found in the “**Layout Rules**” section of the **Manual for designers**.

2. My components merged into a **single image**, and I want to separate them.
My components consist of **several images**, and I want to combine them into one.
You will find the solution to this problem in the **“Naming and tags”** section of the **Manual for designers**.

3. "**Either this file doesn't exist or you don't have permission to view it. Ask the file owner to verify the link and/or update permissions**".
If you see this error, you need to read section "**Teamwork**" in the "**Manual for designers.pdf**".
"**Teamwork**" section of the designer guide will **help you if all the images** in the imported frame **are missing**.

4. If you see these errors, you may have reached your API request limit.



You can reach the limit on Figma API requests, which will prevent you from importing your frames for a while.
To avoid this, follow these guidelines:
• Don't import **more than 100** frames at a time;

If you've **reached** the **limit,** you'll **need to wait** a while to be able to import frames again, or create a new project.
These are **not requirements**, **but recommendations** that **based on personal experience.**

Import Issues

5 ArgumentOutOfRangeException
DA_Assets.FCU.CurrentProject.TryGetByIndex (at CurrentProject.cs)

To resolve this issue, you need to install the correct version of Json.NET.
Information on installing Json.NET through the Package Manager can be found at the beginning of this manual.