Unity 2023 Support

Status	Up-to date

2023 support quick info

Advantages

- It allows to use regular sprites generated by unity in SDFImage
- Allows to get some sprites metadata at runtime

Disadvantages

- Don’t work for decoupled pipeline source sprites
- Works only in Unity 2023

Intro

There is another way to set sprite in unity 2023 with the introduction of Scriptable Objects for Sprites API. This way you can extract `SDFSpriteMetadataAsset` directly from sprite.

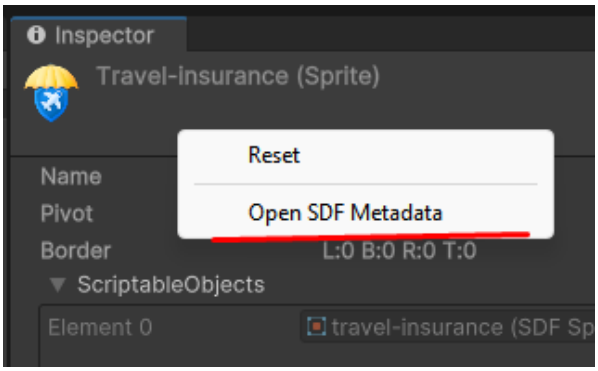
SDF generation produces some data that need to be used by `SDFImage`

- `Sprite` SDFSprite
- `Vector4` BorderOffset
- There might be more data in later versions

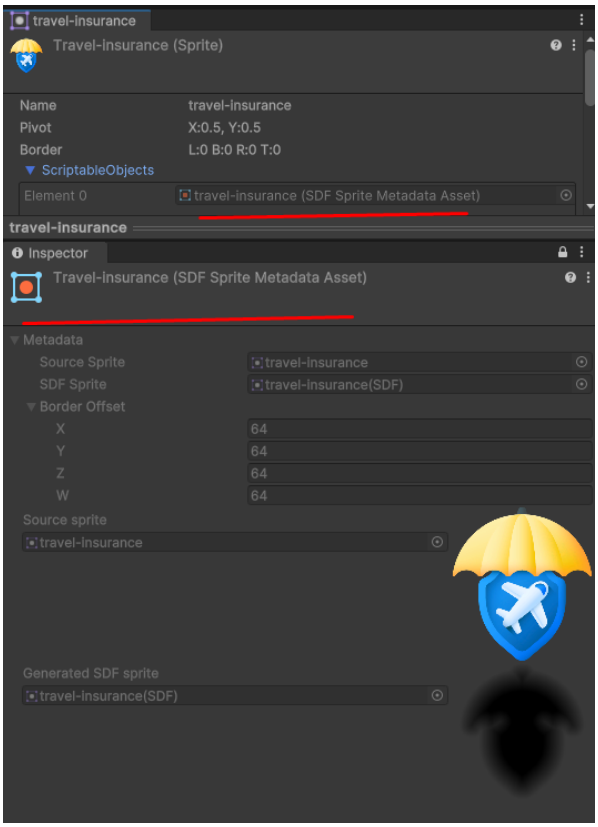
`SDFSpriteMetadataAsset`

All this data is stored in generated `SDFSpriteMetadataAsset`'s, and it unity 2023 its possible to add reference to this asset to sprite directly.

In editor you can see that particular sprite has `SDFSpriteMetadataAsset` that holds all metadata. This asset is hidden in editor for regular pipeline but visible in decoupled.



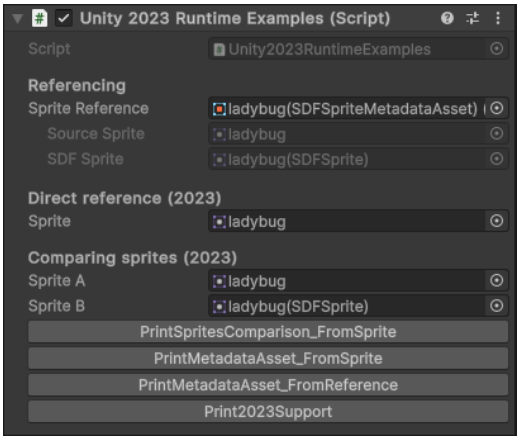
Metadata asset can be selected with context menu.



Meta asset associated with sprite

Samples

Sample script is using all the functions from sections mentioned below



Editor view

```
public void PrintSpritesComparison_FromSprite()
{
#if UNITY_2023_1_OR_NEWER
        var isASdf = _spriteA.IsGeneratedSDFSprite();
        var isASource = _spriteA.IsSourceSDFSprite();

        var isBSdf = _spriteB.IsGeneratedSDFSprite();
        var isBSource = _spriteB.IsSourceSDFSprite();

        var isPair = _spriteA.IsSDFPair(_spriteB);

        Debug.Log($"Sprite: {_spriteA}\n    is SDF: {isASdf}\n    is SDF source: {isASource}\n" +
                  $"Sprite: {_spriteB}\n    is SDF: {isBSdf}\n    is SDF source: {isBSource}\n" +
                  $"Is sprite an SDF pair: {isPair}");
#else
        Debug.LogError($"Unity before 2023 can't get sdf metadata directly from sprite, therefore can't compare sprites");
#endif
    }
```

Performing sprites comparison

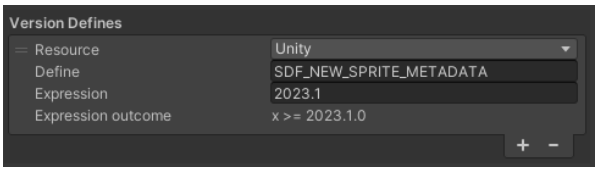
```
    [SampleButton]
    public void PrintMetadataAsset_FromSprite()
    {
#if UNITY_2023_1_OR_NEWER
        var found = _sprite.TryGetSpriteMetadata(out var metadata);
        Debug.Log($"Sprite {_sprite}, metadata found: {found}, metadata: {metadata}");
#else
        Debug.LogError($"Unity before 2023 can't get sdf metadata directly from sprite");
#endif
    }
```

Getting meta asset

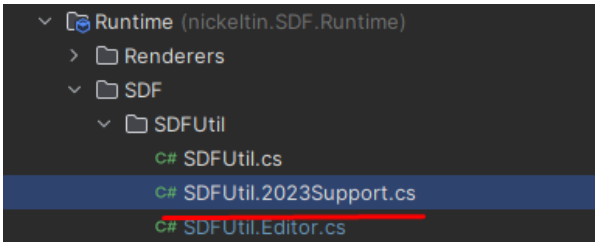
Checkout Unity2023RuntimeExamples.cs to see usages of 2023's SDFUtil

Conditional compilation

To use 2023 features `UNITY_2023_1_OR_NEWER` define can be used. Internally *com.nickeltin.sdf* uses `SDF_NEW_SPRITE_METADATA` defined in *.asmdef



2023 runtime extension methods comes from SDFUtil.2023Support.cs which is conditionally compiled.



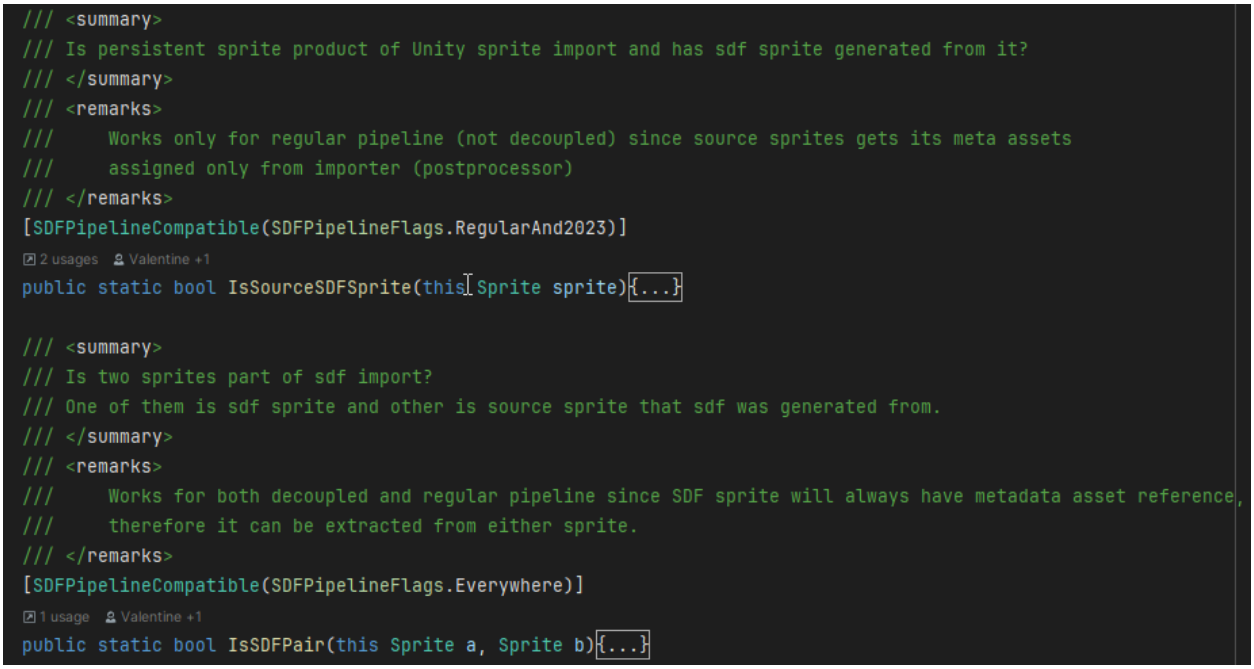
2023 Functions

SDFUtil provides some 2023 only functions

- `TryGetSpriteMetadataAsset` - tries to locate sprite sdf meta asset
- `IsGeneratedSDFSprite` - true if generated sprite is sdf sprite
- `IsSourceSDFSprite` - true if sprite is source unity sprite, and it has sdf sprite generated from it (regular pipeline only)
- `IsSDFPair` - true is one of the sprites is source sprite and other is sdf sprite generated from it

The most important of which is `TryGetSpriteMetadataAsset(this Sprite sprite, out SDFSpriteMetadataAsset metadataAsset)`

```
/// <summary>
/// Will try to get sprite <see cref="SDFSpriteMetadataAsset"/> from sprite.
/// in unity 2023 it will have meta asset assigned in <see cref="Sprite.GetScriptableObjects"/> array.
/// </summary>
/// <remarks>
///     Returns true (and outputs meta asset) if sprite imported from regular pipeline (not decoupled) and has proper sdf import settings.
///     For decoupled pipeline only works if called for SDF sprite.
/// </remarks>
[SDFPipelineCompatible(flags: SDFPipelineFlags.RegularAnd2023 | SDFPipelineFlags.DecoupledSDFSprite)]
7 usages  Valentine +1
public static bool TryGetSpriteMetadataAsset(this Sprite sprite, out SDFSpriteMetadataAsset metadataAsset){...}

/// <summary>
/// Is persistent sprite is product of sdf import?
/// </summary>
/// <remarks>
///     Works for both decoupled and regular pipeline since SDF sprite will always have metadata asset reference,
///     therefore it can be extracted from either sprite.
/// </remarks>
[SDFPipelineCompatible(SDFPipelineFlags.Everywhere)]
2 usages  Valentine +1
public static bool IsGeneratedSDFSprite(this Sprite sprite){...}
```



```
/// <summary>
/// Is persistent sprite product of Unity sprite import and has sdf sprite generated from it?
/// </summary>
/// <remarks>
///     Works only for regular pipeline (not decoupled) since source sprites gets its meta assets
///     assigned only from importer (postprocessor)
/// </remarks>
[SDFPipelineCompatible(SDFPipelineFlags.RegularAnd2023)]
2 usages  Valentine +1
public static bool IsSourceSDFSprite(this Sprite sprite){...}

/// <summary>
/// Is two sprites part of sdf import?
/// One of them is sdf sprite and other is source sprite that sdf was generated from.
/// </summary>
/// <remarks>
///     Works for both decoupled and regular pipeline since SDF sprite will always have metadata asset reference,
///     therefore it can be extracted from either sprite.
/// </remarks>
[SDFPipelineCompatible(SDFPipelineFlags.Everywhere)]
1 usage  Valentine +1
public static bool IsSDFPair(this Sprite a, Sprite b){...}
```

How to use same functions in all versions, not only 2023

`SDFEditorUtilExamples.cs` contains re-creation of runtime samples functionality but for editor, that works in every unity version.

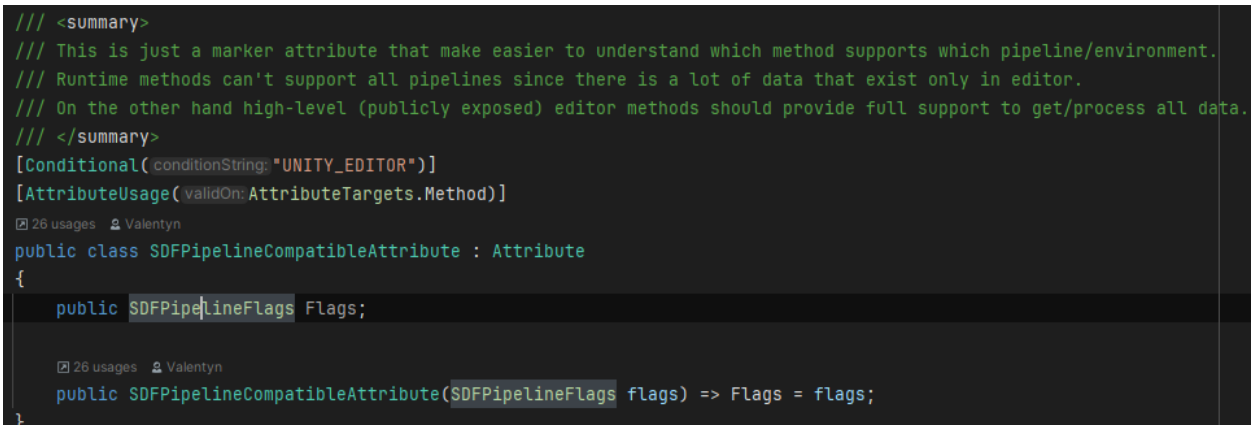
```
[SampleButton]
public void PrintMetadataAsset()
{
    var found :bool = SDFEditorUtil.TryGetSpriteMetadataAsset(_sprite, searchForSDFAsset: false, out var metadata);
    Debug.Log( message: $"Sprite {_sprite}, metadata found: {found}, metadata: {metadata}");
}

[SampleButton]
public void PrintSpritesComparison()
{
    var isASdf = SDFEditorUtil.IsSDFSprite(_spriteA);
    var isASource = SDFEditorUtil.IsSourceSprite(_spriteA, searchForSDFAsset: false);

    var isBSdf = SDFEditorUtil.IsSDFSprite(_spriteB);
    var isBSource = SDFEditorUtil.IsSourceSprite(_spriteB, searchForSDFAsset: false);

    var isPair = SDFEditorUtil.IsSDFPair(_spriteA, _spriteB);

    Debug.Log( message: $"Sprite: {_spriteA}\n    is SDF: {isASdf}\n    is SDF source: {isASource}\n" +
              $"Sprite: {_spriteB}\n    is SDF: {isBSdf}\n    is SDF source: {isBSource}\n" +
              $"Is sprite an SDF pair: {isPair}");

}
```

It just uses SDFEditorUtil, which has the same functions. However editor util is cool because it is compatible with all supported unity versions and all import pipelines.

> 💡 **Checkout SDFEditorUtil for more useful functions!**

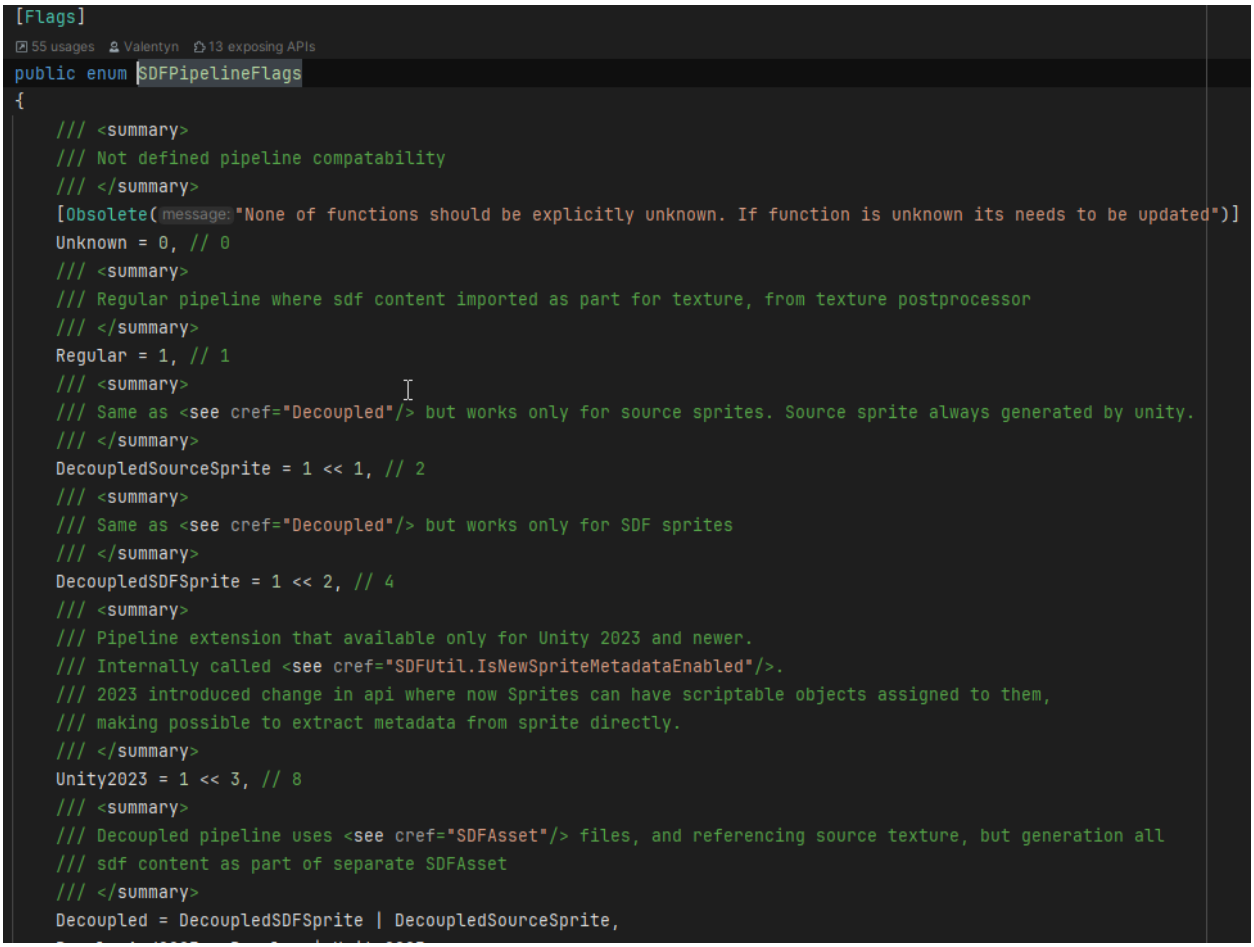
`SDFPipelineCompatibleAttribute`

All utility functions uses this attribute, its just an indicator for the programmer on conditions where this function will work.



```
/// <summary>
/// This is just a marker attribute that make easier to understand which method supports which pipeline/environment.
/// Runtime methods can't support all pipelines since there is a lot of data that exist only in editor.
/// On the other hand high-level (publicly exposed) editor methods should provide full support to get/process all data.
/// </summary>
[Conditional(conditionString: "UNITY_EDITOR")]
[AttributeUsage(validOn: AttributeTargets.Method)]
26 usages  Valentyn
public class SDFPipelineCompatibleAttribute : Attribute
{
    public SDFPipelineFlags Flags;

    26 usages  Valentyn
    public SDFPipelineCompatibleAttribute(SDFPipelineFlags flags) => Flags = flags;
}
```

Not all utility functions is working for all cases, this attribute makes easier to determine which function will work in which conditions.



```
[Flags]
55 usages  Valentyn  13 exposing APIs
public enum SDFPipelineFlags
{
    /// <summary>
    /// Not defined pipeline compatability
    /// </summary>
    [Obsolete(message: "None of functions should be explicitly unknown. If function is unknown its needs to be updated")]
    Unknown = 0, // 0
    /// <summary>
    /// Regular pipeline where sdf content imported as part for texture, from texture postprocessor
    /// </summary>
    Regular = 1, // 1
    /// <summary>
    /// Same as <see cref="Decoupled"/> but works only for source sprites. Source sprite always generated by unity.
    /// </summary>
    DecoupledSourceSprite = 1 << 1, // 2
    /// <summary>
    /// Same as <see cref="Decoupled"/> but works only for SDF sprites
    /// </summary>
    DecoupledSDFSprite = 1 << 2, // 4
    /// <summary>
    /// Pipeline extension that available only for Unity 2023 and newer.
    /// Internally called <see cref="SDFUtil.IsNewSpriteMetadataEnabled"/>.
    /// 2023 introduced change in api where now Sprites can have scriptable objects assigned to them,
    /// making possible to extract metadata from sprite directly.
    /// </summary>
    Unity2023 = 1 << 3, // 8
    /// <summary>
    /// Decoupled pipeline uses <see cref="SDFAsset"/> files, and referencing source texture, but generation all
    /// sdf content as part of separate SDFAsset
    /// </summary>
    Decoupled = DecoupledSDFSprite | DecoupledSourceSprite,
```

`SDFPipelineFlags` is documented as well as all functions that uses the attribute, in which conditions they are working