STL DYNAMICS

DEVELOPER GUIDE

Human Shaders - Runtime API Guide

Runtime API

Runtime namespace:

```
using STLDynamics.HumanShader;
```

Human Shaders separates two kinds of runtime control:

1. Material response - profiles and material utility classes.
2. Skin visibility - controller, paint profiles, visibility sets, requesters, and hide zones.

Material Profiles

Each material family has a ScriptableObject profile with an `ApplyTo(Material)` method.

Available profiles:

```
HumanSkinProfile
HumanHairProfile
HumanEyeProfile
HumanTongueProfile
HumanTeethProfile
HumanClothingProfile
```

Example:

```
using STLDynamics.HumanShader;
using UnityEngine;

public class ApplySkinProfile : MonoBehaviour
{
public HumanSkinProfile profile;
public Material skinMaterial;

public void Apply()
{
if (profile != null && skinMaterial != null)
profile.ApplyTo(skinMaterial);
}
}
```

Each profile validates that the material uses the matching Human Shader before applying values.

Material Utility Classes

Available utility classes:

```
HumanSkinMaterialUtility
HumanHairMaterialUtility
HumanEyeMaterialUtility
HumanTongueMaterialUtility
HumanTeethMaterialUtility
HumanClothingMaterialUtility
```

Each utility exposes a `ShaderName` constant and a material-type check.

Example:

```
if (HumanSkinMaterialUtility.IsHumanSkinMaterial(material))
{
// Safe to treat as a Human Skin material.
}
```

Synchronizing Runtime Material Changes

Some Human Shader properties drive shader keywords and render state.

After changing feature properties directly through code, call the corresponding `Synchronize()` method.

```
material.SetFloat("_EnableTransmission", 1f);
HumanSkinMaterialUtility.Synchronize(material);
```

Hair example:

```
material.SetFloat("_FlowStrength", 1f);
material.SetTexture("_FlowMap", flowMap);
HumanHairMaterialUtility.Synchronize(material);
```

Clothing example:

```
material.SetFloat("_EnableWetness", 1f);
material.SetFloat("_Wetness", 0.8f);
HumanClothingMaterialUtility.Synchronize(material);
```

Synchronization updates the appropriate keywords and, where required, render queue, blending, ZWrite, alpha clipping, and shader passes.

It does not replace your textures or authored material values.

Clothing Material Presets

`HumanClothingMaterialUtility.ApplyMaterialTypePreset()` applies optical defaults while preserving assigned textures and tint.

```
HumanClothingMaterialUtility.ApplyMaterialTypePreset(
clothingMaterial,
HumanClothingMaterialType.Leather);
```

Available material types:

```
HumanClothingMaterialType.Woven
HumanClothingMaterialType.Knit
HumanClothingMaterialType.Denim
HumanClothingMaterialType.SatinSilk
HumanClothingMaterialType.Velvet
HumanClothingMaterialType.Leather
HumanClothingMaterialType.Synthetic
```

Skin Visibility: Direct Profile Toggle

```
public HumanSkinVisibilityController controller;
public HumanSkinPaintProfile torsoProfile;

public void SetTorsoHidden(bool hidden)
{
controller.SetProfileHidden(torsoProfile, hidden);
}
```

Query current state:

```
bool hidden = controller.IsProfileHidden(torsoProfile);
```

Add or remove a profile from the controller:

```
controller.AddProfile(torsoProfile, hiddenAtStart: false);
controller.RemoveProfile(torsoProfile);
```

Skin Visibility: String Region ID

A paint profile has a persistent runtime Region ID.

If that profile is already registered on the controller:

```
bool found = controller.SetProfileHidden(regionId, true);
```

Use unique IDs. Human Shaders Diagnostics warns if duplicated profile assets share the same ID.

Skin Visibility: Visibility Sets

For runtime equipment systems, visibility sets are normally cleaner than manually toggling profiles.

```
public HumanSkinVisibilityController controller;
public HumanSkinVisibilitySet shirtVisibility;

private int shirtToken;

public void EquipShirt()
{
shirtToken = controller.ApplyVisibilitySet(shirtVisibility);
}

public void UnequipShirt()
{
if (shirtToken != 0)
{
controller.RemoveVisibilityRequest(shirtToken);
shirtToken = 0;
}
}
```

Important

Keep the token returned by `ApplyVisibilitySet()` and release that same token.

The request system is reference counted so overlapping garments can safely request the same painted region.

Skin Visibility: Requester Component

A `HumanSkinVisibilityRequester` automates the token lifecycle.

Typical clothing setup:

```
Shirt GameObject
└── HumanSkinVisibilityRequester
├── Visibility Set = Shirt_Hide
└── Apply On Enable = true
```

When the GameObject becomes enabled, the requester calls `Apply()`.

When disabled, it calls `Release()`.

You can also call these explicitly:

```
requester.Apply();
requester.Release();
```

Skin Visibility: Hide Zones

A zone can be enabled or disabled directly:

```
zone.SetHidden(true);
zone.SetHidden(false);
```

Change shape:

```
zone.Shape = HumanSkinHideZoneShape.Box;
```

Available shapes:

```
HumanSkinHideZoneShape.Sphere
HumanSkinHideZoneShape.Box
HumanSkinHideZoneShape.Capsule
HumanSkinHideZoneShape.CustomConvexMesh
```

Primitive values can be modified at runtime:

```
zone.Radius = 0.25f;
zone.Height = 1.0f;
zone.BoxSize = new Vector3(0.5f, 1.0f, 0.3f);
```

A custom convex mesh can be assigned:

```
zone.CustomConvexMesh = myConvexMesh;
```

Refreshing Visibility

Most runtime changes are applied automatically while the controller is active.

To explicitly force a rebuild:

```
controller.RefreshVisibility();
```

For custom systems that changed paint data or zone data:

```
controller.MarkPaintDirty();
controller.MarkZonesDirty();
```

Assigning Target Renderers from Code

```
controller.SetTargetRenderers(new Renderer[]
{
bodyRenderer,
headRenderer
});
```

Only renderers that use the Human Skin shader will participate meaningfully in the visibility shader workflow.

Runtime Wetness Example

Clothing:

```
using STLDynamics.HumanShader;
using UnityEngine;

public class ClothingWetnessExample : MonoBehaviour
{
public Material clothingMaterial;

public void SetWetness(float amount)
{
if (!HumanClothingMaterialUtility.IsHumanClothingMaterial(clothingMaterial))
return;

clothingMaterial.SetFloat("_EnableWetness", 1f);
clothingMaterial.SetFloat("_Wetness", Mathf.Clamp01(amount));
HumanClothingMaterialUtility.Synchronize(clothingMaterial);
}
}
```

Runtime Eye Wetness Example

```
using STLDynamics.HumanShader;
using UnityEngine;

public class EyeWetnessExample : MonoBehaviour
{
public Material eyeMaterial;

public void SetWetness(float amount)
{
if (!HumanEyeMaterialUtility.IsHumanEyeMaterial(eyeMaterial))
return;

eyeMaterial.SetFloat("_Wetness", Mathf.Clamp01(amount));
HumanEyeMaterialUtility.Synchronize(eyeMaterial);
}
}
```

Material Instance Warning

Changing a shared material asset changes every renderer using that material.

When runtime values need to differ per character, use a runtime material instance or your own material-management strategy.

For per-character skin visibility, Human Shaders uses `MaterialPropertyBlock` internally through `HumanSkinVisibilityController`, so visibility does not require duplicating the skin material simply to hide different regions.