Asset Registry

How assets are discovered by the editor and how to make it know more about asset types before they are loaded.

The **Asset Registry** is an editor subsystem which gathers information about unloaded assets asynchronously as the editor loads. This information is stored in memory so the editor can create lists of assets without loading them. The information is authoritative and is kept up to date automatically as assets are changed in memory or files are changed on disk. The **Content Browser** is the primary consumer for this system, but it may be used anywhere in editor code.

Obtaining a List of Assets

To form a list of assets by class, simply load the Asset Registry module then invoke `Module.Get().GetAssetsByClass()`

```
FAssetRegistryModule& AssetRegistryModule = FModuleManager::LoadModuleChecked<FAssetRegistryModule>("AssetRegistry");
TArray<FAssetData> AssetData;
const UClass* Class = UStaticMesh::StaticClass();
AssetRegistryModule.Get().GetAssetsByClass(Class->GetFName(), AssetData);

```

Copy full snippet

This will return a list of `FAssetData` objects which describe assets that may be loaded or unloaded. `FAssetData` objects hold information about an asset that can be determined before it is loaded.

Here is a list of its members and descriptions:

Member	Description
`FName ObjectPath`	The object path for the asset in the form `Package.GroupNames.AssetName`.
`FName PackageName`	The name of the package in which the asset is found.
`FName PackagePath`	The path to the package in which the asset is found.
`FName GroupNames`	The '`.`' delimited list of group names in which the asset is found. `NAME_None` if there were no groups.
`FName AssetName`	The name of the asset without the package or groups.
`FName AssetClass`	The name of the asset's class.
`TMap<FName, FString> TagsAndValues`	The map of values for properties that were marked `AssetRegistrySearchable`. See Tags and Values for more information.

You can also form lists using other criteria by invoking one of the following functions:

Function	Description
`GetAssetsByPackageName()`	Returns a list of assets from a particular package.
`GetAssetsByPath()`	Returns a list of assets in a specified path.

Function	Description
`GetAssetByObjectPath()`	Returns a list of assets with the specified object paths.
`GetAssetsByTagValues()`	Returns a list of assets with the specified set of tags and values.
`GetAllAssets()`	Returns a list of all assets. This can be slow.

If you need to form a list of assets using multiple criteria, use `GetAssets()` and supply a `FARFilter` struct as described in the Creating a Filter section.

Converting FAssetData to UObject*

`FAssetData` objects have a function named `GetAsset()` which will return the `UObject*` that the `FAssetData` represents. This will load the asset if needed then return it.

If you just want to check if an asset is loaded, use `IsAssetLoaded()` instead.

Creating a Filter

A `FARFilter` can be supplied when invoking `GetAssets()` to create a list of assets which are filtered down by multiple criteria. A filter is comprised of multiple components:

- PackageName
- PackagePath
- Collection
- Class
- Tags/Value pairs

A component may have more than one element. An asset passes a filter if it satisfies **ALL** components. To satisfy a component, an asset must match **ANY** of the elements within.

For example, if a StaticMesh asset exists whose path is /Game/Meshes/BeachBall:

- The asset will pass if the filter includes only the PackagePath `/Game/Meshes`. There is only one component which has one element.
- The asset will pass if the filter includes the PackagePath `/Game/Meshes` and Classes `UParticleSystem` **AND** `UStaticMesh`. There are two components where the first has one element and the second has two.
- The asset will fail if the filter includes the PackagePath `/Game/Meshes` and **ONLY** the Class `UParticleSystem`. There are two components that each have one element.
- The asset will fail if the filter includes the PackagePath `/Game/NotMeshes` and the Class `UStaticMesh`. This filter also uses two components that each have one element.

An example of using a filter with two components, Class and PackagePath:

```
FAssetRegistryModule& AssetRegistryModule =
FModuleManager::LoadModuleChecked<FAssetRegistryModule>("AssetRegistry");
TArray<FAssetData> AssetData;
FARFilter Filter;
Filter.Classes.Add(UStaticMesh::StaticClass());
Filter.PackagePaths.Add("/Game/Meshes");
AssetRegistryModule.Get().GetAssets(Filter, AssetData);

```

Copy full snippet

Tags and Values

`FAssetData` objects returned from the Asset Registry contain a name and value map called `TagsAndValues`. This is a list of property names and associated values for the asset the `FAssetData` represents. This information is gathered when an asset is saved and stored in the header of the `UAsset` file that contains the asset. The Asset Registry reads this header and fills out the `TagsAndValues` map accordingly. The Asset Registry only gathers properties that are marked with the `AssetRegistrySearchable` `UPROPERTY()` flag.

For example (from `UTexture`):

```
/** The texture filtering mode to use when sampling this texture. */
UPROPERTY(Category=Texture, AssetRegistrySearchable)
TEnumAsByte<enum TextureFilter> Filter;
```

```
```

Copy full snippet

Once this flag was added to the Filter property of `UTexture`, all `UTextures` that were saved thereafter now have an entry in their `FAssetData`'s `TagsAndValues` map whose key is `Filter` and whose value is the string representation of the enum value, such as `"TF_Linear"`.

This requires assets to be resaved before their properties will be discovered by the Asset Registry.

If you want the Asset Registry to be able to search for information that is not directly a UProperty, your asset's class can implement the virtual function: GetAssetRegistryTags() to manually add key/value pairs to the TagsAndValues map. GetAssetRegistryTags is inherited from UObject.

Asynchronous Data Gathering

The Asset Registry reads `UAsset` files asynchronously and may not have a complete list of all assets at the time that you request it. If your editor code requires a complete list, the Asset Registry provides delegate callbacks for when assets are discovered/created, renamed, or removed. There is also a delegate for when the Asset Registry has completed its initial search which is useful for many systems.

You may sign up for these delegates by loading the Asset Registry module, then using the functions provided in `IAssetRegistry`:

```
/** Register/Unregister a callback for when assets are added to the registry */
virtual FAssetAddedEvent& OnAssetAdded() = 0;

/** Register/Unregister a callback for when assets are removed from the registry */
virtual FAssetRemovedEvent& OnAssetRemoved() = 0;

```

```
/** Register/Unregister a callback for when assets are renamed in the registry */
virtual FAssetRenamedEvent& OnAssetRenamed() = 0;

/** Register/Unregister a callback for when the asset registry is done loading files */
virtual FFilesLoadedEvent& OnFilesLoaded() = 0;

/** Register/Unregister a callback to update the progress of the background file load */
virtual FFileLoadProgressUpdatedEvent& OnFileLoadProgressUpdated() = 0;

/** Returns true if the asset registry is currently loading files and does not yet know about all assets */
virtual bool IsLoadingAssets() = 0;

```

Copy full snippet

For example:

```
void FMyClass::FMyClass()
{
// Load the asset registry module to listen for updates
FAssetRegistryModule& AssetRegistryModule = FModuleManager::LoadModuleChecked<FAssetRegistryModule>("AssetRegistry");
AssetRegistryModule.Get().OnAssetAdded().AddRaw( this, &FMyClass::OnAssetAdded );
}

FMyClass::~FMyClass()
{
// Load the asset registry module to unregister delegates
FAssetRegistryModule& AssetRegistryModule = FModuleManager::LoadModuleChecked<FAssetRegistryModule>("AssetRegistry");
AssetRegistryModule.Get().OnAssetAdded().RemoveAll( this );
}

void FMyClass::OnAssetAdded(const FAssetData& AssetData)
{
// An asset was discovered by the asset registry.
// This means it was either just created or recently found on disk.
// Make sure code in this function is fast or it will slow down the gathering process.
}

```

Copy full snippet

The Asset Registry can be used in commandlets, but gathers information synchronously instead. The `LoadModule()` call will block until the gather is complete.

If your code waits for assets to be discovered asynchronously and has a Slate UI frontend, it should contain an `SAssetDiscoveryIndicator` widget to convey progress to the user.