# Data Validation

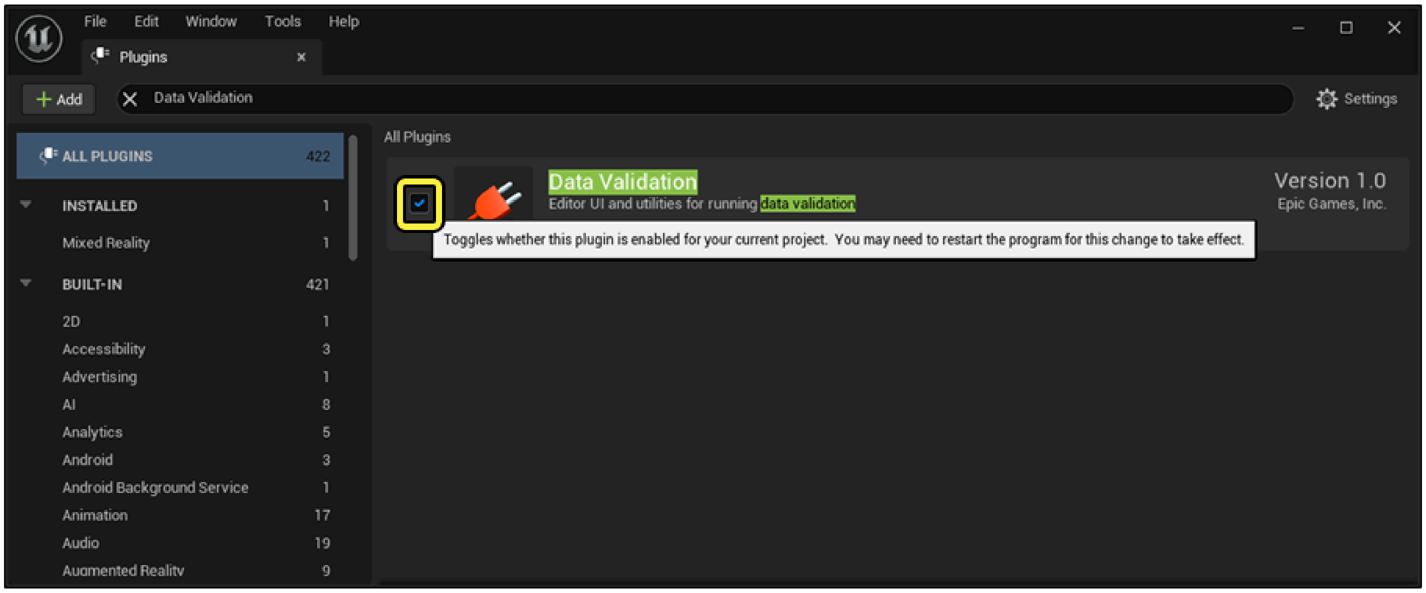
Developers can extend this system to validate assets with custom-scripted rulesets.

The **Unreal Engine (UE) Editor** features a **Data Validation** plugin for developers to validate assets with custom-scripted rulesets. Common validation use cases include:

- Checking that assets meet name conventions
- Enforcing space and performance budgets
- Catching non-cyclic dependencies

## Plugin Verification

To verify that the built-in Data Validation plugin is enabled by default, from the main menu, select **Edit** > **Plugins**, and from the Plugins menu, search for **Data Validation**.



Editor UI and utilities for running data validation

Toggles whether this plugin is enabled for your current project. You may need to restart the program for this change to take effect.

## Editor Usage

Developers can use the Data Validation system in many ways, from testing a single asset to validating all of a project's assets.

| Validation Use Case | Usage | Notes |
|---|---|---|
| Asset | In the **Content Browser**, right-click an asset and select **Asset Actions** > **Validate Assets** | Validates a specific asset; however, it is possible to select multiple assets for validation. |
| Assets and dependencies | In the **Content Browser**, right-click an asset and select **Asset Actions** > **Validate Assets and Dependencies** | Validates a specific asset and its dependencies; however, it is possible to validate multiple assets at once. |
| Folder's assets | In the **Content Browser**, right-click a folder and select **Validate Assets in Folder** | Validates a specific folder; however, it is possible to validate multiple folders at once. |
| Project's assets | From the main menu, select **Tools > Validate Data...** | Validates all of the assets in a project's content directory. |

## Command Line Usage

Running the plugin from the command line is useful to developers who want to validate assets as part of a **Continuous Integration System (CIS)**. Perform command line validation with the following commandlet:

`UnrealEditor-Cmd.exe <PROJECT_NAME>.uproject -run=DataValidation`

- The Data Validation system only runs C++ validation rules by default.
- Developers can extend the Data Validation system to support Blueprint and Python validation rules.

## Validation Rules

Currently, there are two ways to create validation rules:

- Have a custom UObject-derived class that overrides `IsDataValid`. This works best for custom classes for your project.
    - This method provides access private/protected class data and functions that the `UEditorValidatorBase` method would not.
    - Because you are implementing your own class in this case, you can also set up a `BlueprintImplementableEvent` to be called by `IsDataValid`, so you can have some validation logic in C++ and further validation logic in Blueprints or Python.
- Creating a `UEditorValidatorBase` derived class using C++, Blueprints, or Python. The two key functions here are `CanValidateAsset` and `ValidateLoadedAsset`, both of which take a `UObject` pointer as an argument to validate. This system lets you validate any asset, whether it is based on a default engine class or a custom class.
    - `ValidateLoadedAsset` must call `AssetPasses` or `AssetFails` for each of its execution paths.
    - Validators can be enabled by overriding `IsEnabled`. By default, this returns the protected variable `bIsEnabled` which is exposed to **Details** panels as well.
    - C++ and Blueprint validators will be discovered automatically on editor startup, while Python validators will need to register themselves with the `UEditorValidatorSubsystem` using `AddValidator`.

- By default, Blueprint validator auto-registration is disabled in **Fortnite** for performance reasons.

Both types of validation are run by CIS, on asset save (enabled by default), and through menu options in the editor and Content Browser. When validation fails, messages will be displayed in the Message Log in the editor, and in the console in CIS.

If you are implementing your own calls to `UEditorValidatorSubsystem`, you will need to handle the display of the `ValidationErrors` `FText` array yourself. The functions needed to validate an asset from its `FAssetData` or the loaded `UObject` are available to scripting as well.

This section references these APIs, read them to learn more:

- IsDataValid
- DataValidation