Developer

- / Documentation
- / Unreal Engine ∨
- / Unreal Engine 5.4 Documentation
- / Programming and Scripting
- / Unreal Architecture
- / String Handling
- / FName

#### **FName**

Reference for creating, converting, and comparing FNames in Unreal Engine.



FNames are a lightweight type used for efficient string handling. Specifically, Unreal Engine maintains a global table of unique strings, and an FName stores an instance number and an index reference to a given string, facilitating quick lookup and access.

Additionally, the FName subsystem uses a hash table to provide fast string-to-FName conversions.

FNames are particularly useful for representing object names, identifiers, and other strings that are frequently compared. When you name a new asset in the **Content Browser**, change a parameter in a Dynamic Material Instance, or access a bone in a Skeletal Mesh, you are using **FNames**.

FNames are case-insensitive, immutable, and cannot be manipulated.

## **Creating FNames**

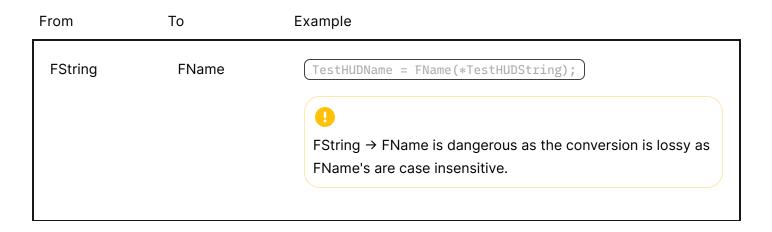
### **Conversions**

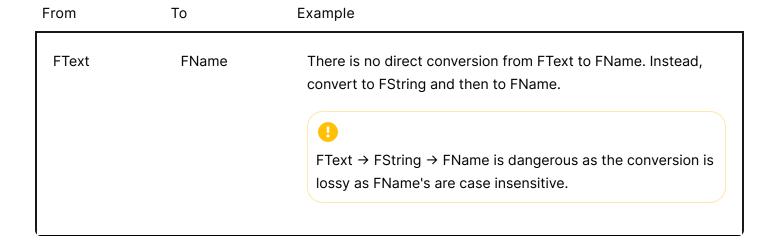
FNames can only be converted to FStrings and FText, and can only be converted from FStrings.

#### **From FName**

| From  | То      | Example  |
|-------|---------|--|
| FName | FString | <pre>TestHUDString = TestHUDName.ToString();</pre>   |
| FName | FText   | <pre>TestHUDText = FText::FromName(TestHUDName);</pre>   |
|       |         | • Properties of the state o |
|       |         | the FNames's content will not benefit from the FText's "auto localization".  |
|       |         |  |

#### **To FName**





When doing these conversions, be aware that they may contain characters that are not valid for the type of FName you are creating. The <a href="INVALID\_NAME\_CHARACTERS">INVALID\_NAME\_CHARACTERS</a> macro in <a href="NameTypes.h">NameTypes.h</a> defines which characters are invalid for FNames, and several functions within FName (such as <a href="IsValid0bjectName">IsValid0bjectName</a> () check the validity of FNames for particular use cases.

## **Comparisons**

The == operator can be used to compare two FNames, and returns true or false. Rather than doing string comparisons, this compares the values in Index, which is a significant CPU savings.

**FName::Compare** can also be used to compare two FNames, and it will return less than 0 if this less than 0ther, 0 if this equal to 0ther, and greater than 0 if this greater than 0ther.

```
1 CompareFloat = TestFName.Compare(OtherFName);
2
Copy full snippet
```

# **Using FNames**

Using an FName is pretty straightforward. Let us say that you wanted to get the bone named pelvis from the Skeletal Mesh Component of an Actor. The C++ code below shows the use of

an FName constructed on the fly while being passed to GetBoneRotation()

Copy full snippet

This creates an FName that is passed to GetBoneRotation(), which returns the FRotator for that corresponding bone. The name of the bone gets loaded into the FName table when the package loads, so the constructor for the FName finds the bone's name in the hash table, avoiding an allocation.

## **Searching the Name Table**

If you want to determine if an FName is in the name table, but you do not want to automatically add it, you can supply a different search type to the FName constructor:

```
1 if( FName(TEXT("pelvis"), FNAME_Find) != NAME_None )
2 {
3 // Do something
4 }
```

Copy full snippet

If the name does not exist in the name table, the Index of the FName is set to NAME\_None Note that we are not checking a pointer for null as you would with a normal string.