String Handling

An overview of the string classes available in Unreal with reference guides for FName, FText, and FString.

FName

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.

- FName Reference Guide

FText

In **Unreal Engine (UE)** the primary component for text localization is the `FText` class. All user-facing text should use this class, as it supports text localization by providing the following features:

- Creating localized text literals.
- Formatting Text (to generate text from a placeholder pattern).
- Generating text from numbers.
- Generating text from dates and times.
- Generating derived text, such as making text upper or lower case.

`FText` also features the `AsCultureInvariant` function (or the `INVTEXT` macro), which creates non-localized, or "culture invariant" text. This is useful for things like converting a player name from an external API into something you can display in your user interface.

You can create a blank `FText` using either `FText::GetEmpty()`, or by using just `FText()`.

- FText Reference Guide

FString

Unlike `FName` and `FText`, `FString` can be searched, modified, and compared against other strings. However, these manipulations can make `FStrings` more expensive than the immutable string classes. This is because `FString` objects store their own character arrays, while `FName` and `FText` objects store an index to a shared character array, and can establish equality based purely on this index value.

- FString Reference Guide

Printf

The `FString` function, `Printf`, can create `FString` objects the format argument specifiers as the C++ `printf` function. Similarly, the `UE_LOG` macro prints a `printf` formatted string to the screen, log output, and log files, depending on what type of UE4 build is running.

Remember that to use these strings and conversion you will need to include the necessary header files. A list of which header files are needed can be found on the API reference page for each string.

Conversions

From	To	Example
FName	FString	`TestHUDString = TestHUDName.ToString();`
FName	FText	`TestHUDText = FText::FromName(TestHUDName);` ❗ FName → FText is valid in some cases, but be aware that the FNames's content will not benefit from the FText's "auto localization".
FString	FName	`TestHUDName = FName(*TestHUDString);` ❗ FString → FName is dangerous as the conversion is lossy as FName's are case insensitive.
FString	FText	`TestHUDText = FText::FromString(TestHUDString);` ❗ FString → FText is valid in some cases, but be aware that the FString's content will not benefit from the FText's "auto localization".
FText	FString	`TestHUDString = TestHUDText.ToString();` ❗ FText → FString is dangerous as it is a potentially lossy conversion for some languages.

From	To	Example
FText	FName	There is no direct conversion from FText to FName. Instead, convert to FString and then to FName. FText → FString → FName is dangerous as the conversion is lossy as FName's are case insensitive.
FString	int32	`int32 TestInt = FCString::Atoi(*MyFString);`
FString	float	`float TestFloat = FCString::Atof(*MyFString);`
int32	FString	`FString TestString = FString::FromInt(MyInt);`
float	FString	`FString TestString = FString::SanitizeFloat(MyFloat);`

Encoding

In general, you should be using the **TEXT()** macro when setting string variable literals. If you do not specify the TEXT() macro, your literal will be encoded using ANSI, which is highly limited in what characters it supports. Any ANSI literals being passed into FString need to undergo a conversion to TCHAR (native Unicode encoding), so it is more efficient to use TEXT().

For more about encoding, see the Character Encoding documentation.