StatsTracer Plugin

For Unreal Engine 4

Document-Author	*Tobias Stein*
Document-Version	*1.0 (released 01/04/2018)*

Table of Content

Overview

StatsTracer is an analytical tool which enables you to monitor changes of game stats (e.g. actor properties) in real time (YouTube: https://www.youtube.com/watch?v=mE0ZQu1p9aA).

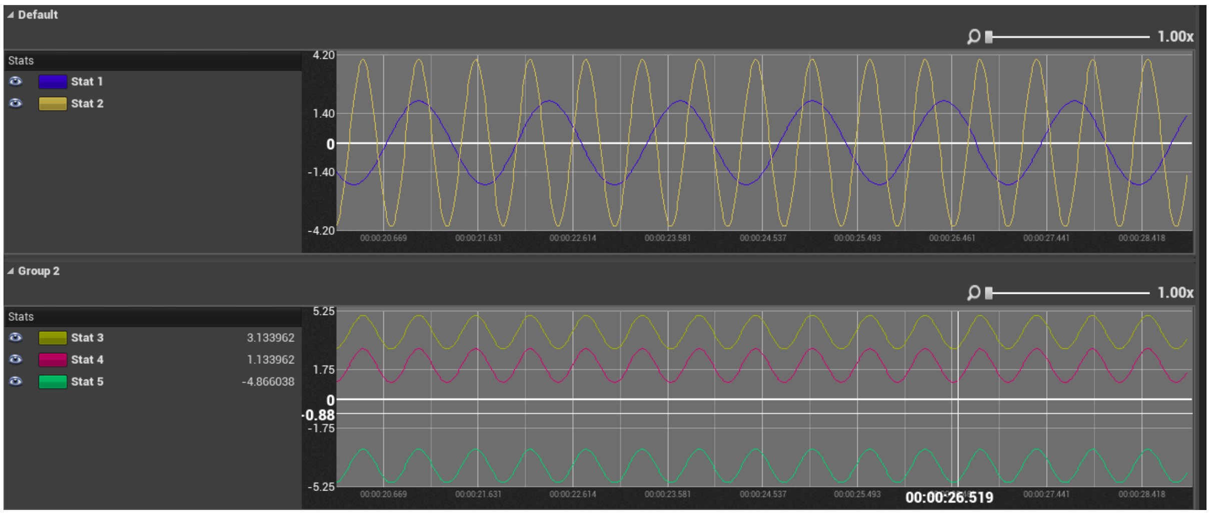


Figure 1: Sample tracer chart plotting five different actor stats.

In this context the term ***stat*** is referring to any game variable that can change over time. Typical game stats are actor properties like health, position etc. Stats are assumed to be any of the following data types: *bool*, *int*, *byte*, *float*, *FVector*, *FRotator* or *FTransform*. So how does it work? In order to get an output presented to you by the plugin you will have to define a **tracer**. A tracer is an object which is correlated to one game object in the (PIE-)world. There are actually three different ways to create a tracer. You can use blueprints, the TracerComponent or C++ code (see the HowTo section for more information). When a new game is started a new tracer-**session** will be created before the "*BeginPlay*" event is send. After that, tracer objects can be created and registered to the active session.

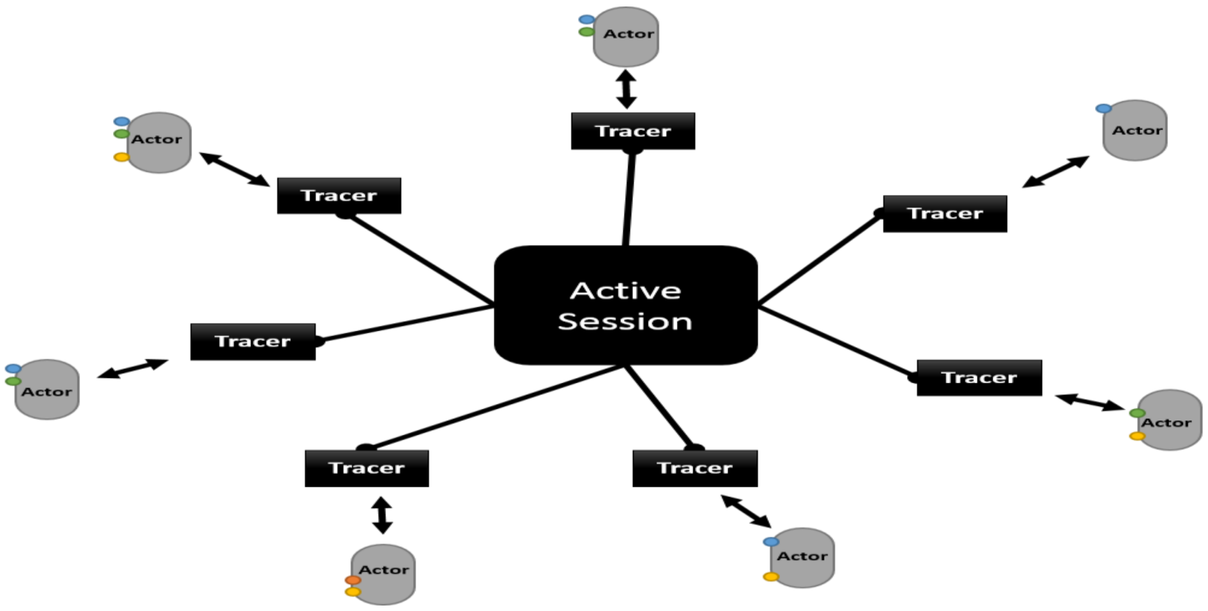


Figure 2: Actor, Tracer and Session.

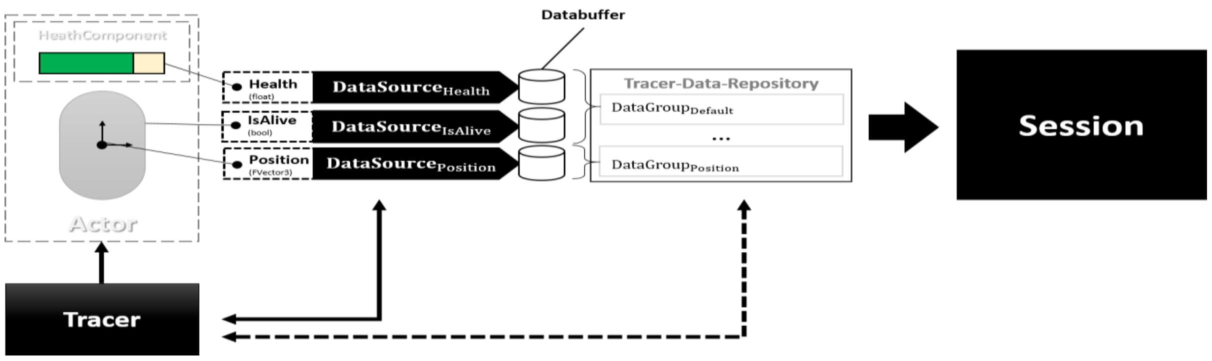


Figure 3: Tracer data repository.

Figure 3: Tracer data repository. provides a closer look how tracers are coupled with objects, in this example an actor with a health-component. First of all, a tracer is no more than a proxy object and does not hold any data samples of the target objects stats. When a new tracer is created it will get a reference to the target object and a few more attributes, like a name for the tracer and description. Internally a global manager will use the reference of the target object to allocate a new data-repository where all the later sampled data will be stored. If a data-repository has been already allocated for the target, this repository will be used again. This might happen if two tracer objects with the same target object reference will be created. Once the tracer object has been created it can be used to add stats of the target object to the data-repository. For each stat added a data-source will be created. These data-sources can be seen as some kind of link between a stat and a data-buffer in the data-repository. they will be frequently sampled to get the current value of a stat at a given time. They further describe how stats are grouped, what color their label should have, a name and description. After all required stats are added, that is, their data-sources have been created the tracer object can be used to start, pause/resume or stop the sampling for the target object. If the active session or game will be terminated all tracer object will be stopped.

Features

Easy access – The plugin is easily accessed via the Unreal Editor toolbar icon or using the menu (Window › Developer Tools › StatsTracer).

Customizable – There are various options in the *Project Settings* panel which can be customized, like chart grid size and display, sampling frequency or memory consumption limits. Data charts can be resized, recolored and zoomed. Stats can be arranged in multiple groups and therefore displayed in separate charts. Groups can be collapsed and single stats can be hidden. Chart panels are free floating windows but can also be docked.

CSV export – Traced stats data can be streamed as csv content to file.

Automatic stat detection – Traceable stats can easily be detected and explored through the *TracerComponent*.

Blueprint functions – Users can manage the life-cycle of StatsTracers from their blueprints. Additional functions allow a subtler and dynamic way to create, pause, resume or stop tracers and to add stats to them.

C++ API – Tracer can be created in C++ code.

Editor Slate UI – All data can be viewed and managed though Slate UI widgets.

Limitations

Development and Editor usage only – This plugin can only be used during the development process in the Unreal Editor.

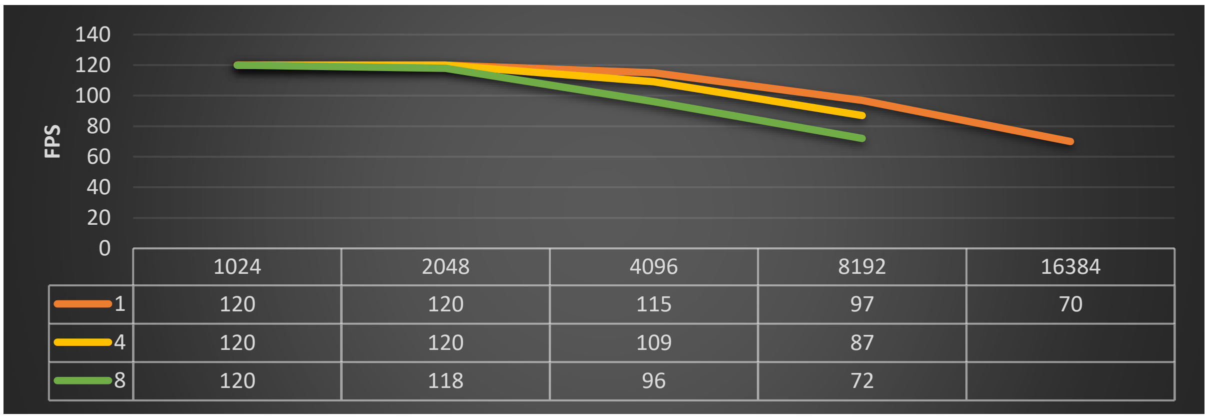
Temporal data storage – Traced stat data is only kept as long as Unreal Editor instance is running or StatsTracer session is available. The amount of temporal data kept in memory depends on available system resources and user settings. If data is required to be persisted for later evaluation the CSV streaming option should be activated for tracers.

Limited sampling buffer – For efficiency reasons an upper limit for sampling buffers must be provided. During an active session data sampled by tracers are stored in internal buffers. Once the upper limit is reached the oldest sample is dropped and the newest is add to the buffer. The buffer size can range from 8 to 16384 and can be reset for every new session without restarting the editor. Streamed CSV content is not affected by this limitation and will contain all data samples.

Datatypes – In total there are seven variable types that you can trace. These types are: bool, int, float, byte, FVector, FRotator and FTransform.

Performance

Using StatsTracers in a project comes with an additional cost of memory and computation overhead during play mode. The following chart shows the average framerate during sessions with different StatsTracer setups.



	1024	2048	4096	8192	16384
🟧 1	120	120	115	97	70
🟨 4	120	120	109	87	
🟩 8	120	118	96	72	

Figure 4: Fps during play mode with different StatsTracer setups.

This benchmark was conducted on a local machine with a Amd Ryzen 1700 (8 cores @3.0GHz) CPU, 16GB RAM and a Nvidia GeForce GTX 1080.

The chart shows the results of 15 different setups. A single setup can be described by the number of tracers (horizonal axis: 1024, 2048 … 19384) and the number of stats sampled by each tracer (1, 4 and 8). As you can see a number of 2000 tracer with an average of 4 sampled stats per tracer (~8000 sampled stats in total) does not affect effect the overall performance of the main loop. By doubling these numbers, the framerate will decrease in an exponential way. Another fact that must be took into account is the memory consumption. Each tracer created and each stat traced will consume memory. The StatsTracer plugin provides a monitor that will tell you the theoretically consumed memory according to the total number of traced stats. However, this value is not accurate as it does not take the engines memory inhouse-bookkeeping overhead into account. While conducting the benchmark with ~65536 total traced stats the total memory consumption exceeded 16GB RAM and was crashing the editor without any exception message.

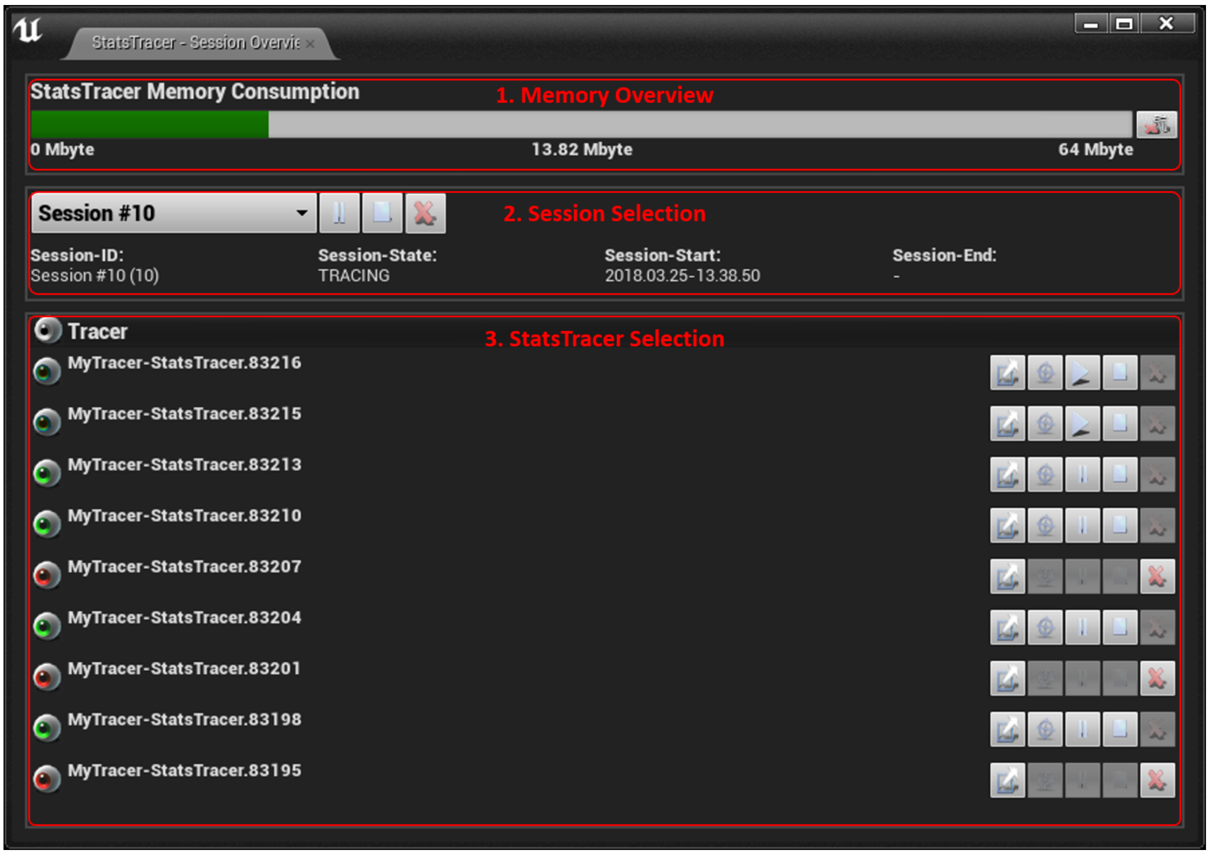
Graphical User Interfaces

This section will provide a full coverage of all UI Widgets used to interact with the StatsTracer plugin.

There are three important Widgets. A main window which will shows up when pressing the icon in the toolbar or when using the menu entry in the editor. The second window is the tracer details window. Last but not least the details panel of the TracerComponent.

StatsTracer Main-Window

The main window is divided in three sections. The first section gives you information about the current memory usage of the StatsTracer plugin. In the second section you will find controls to select or change a tracer session. The third section shows a list of all available tracer belonging to the current selected session.



StatsTracer Memory Consumption

0 Mbyte 13.82 Mbyte 64 Mbyte

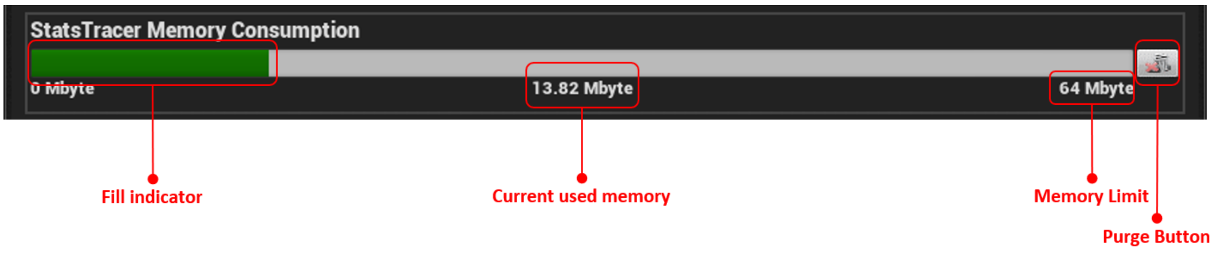
Session #10

Session-ID:	Session-State:	Session-Start:	Session-End:
Session #10 (10)	TRACING	2018.03.25-13.38.50	-

Tracer

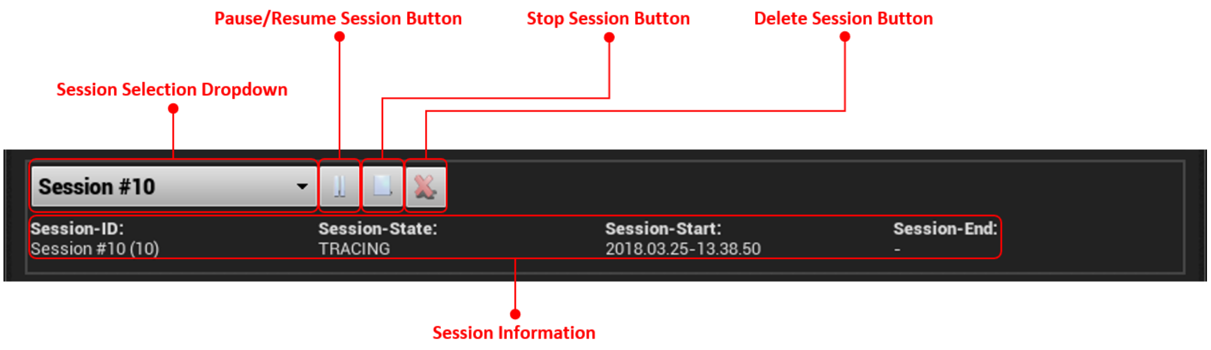
MyTracer-StatsTracer.83216
MyTracer-StatsTracer.83215
MyTracer-StatsTracer.83213
MyTracer-StatsTracer.83210
MyTracer-StatsTracer.83207
MyTracer-StatsTracer.83204
MyTracer-StatsTracer.83201
MyTracer-StatsTracer.83198
MyTracer-StatsTracer.83195

Memory Overview



Label	Description
Fill indicator	An indicator that show the current memory consumption. The indicator will change color from green to red as more and more memory is used.
Current used memory	An approximate value of how much memory is used by the plugin for storing its data. The value is based on all temporarily stored sessions, tracers and their stats. Note that the actual used memory is bigger than this value shown here as the engine used internal book-keeping mechanisms to manage memory.
Memory Limit	The maximum memory allowance for the plugin. If this limit is reached (indicator is totally filled up) no more data will be sampled and further started sessions may remain empty.
Purge Button	Pressing the *Purge Button* will clear all internal stored data (sessions, tracers and their data) freeing all allocated memory. *Note: this button becomes only available (enabled) if not in play-mode.*

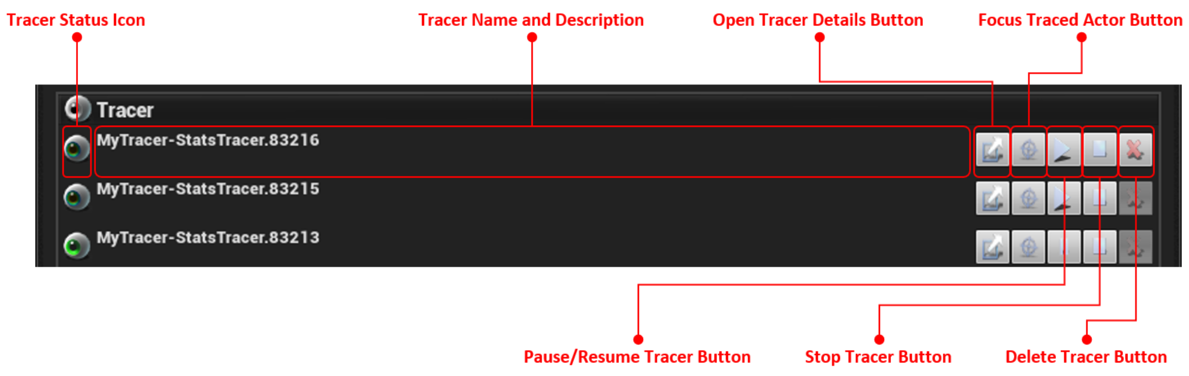
Session Selection



Session-ID: Session #10 (10)
Session-State: TRACING
Session-Start: 2018.03.25-13.38.50
Session-End: -

Label	Description
Session Selection Dropdown	The Session Selection Dropdown will show all currently available sessions. Sessions can be selected or changed by simply open the dropdown and selecting an item.
Pause/Resume Session Button	This toggle will pause/resume the current active session. If a session is paused/resumed all its tracers will be paused/resumed. This won't override the tracers state, that is, if a tracer is paused and a session will be resumed the tracer remains paused. *Note: this button becomes only available (enabled) if current selected session is active and not stopped.*
Stop Session Button	This stops the current active session. Once a session is stopped it cannot reactivated to continue tracing. *Note: this button becomes only available (enabled) if current selected session is active and not stopped.*
Delete Session Button	This will remove the current selected session and all its tracers and their data freeing up the allocated memory. *Note: this button becomes only available (enabled) if current selected session is completed.*
Session Information	This section shows the very basic session information: the sessions id, the current session state (tracing, paused, stopped, complete), the session start-time and end-time. The end-time will be shown once the session has been stopped/completed.

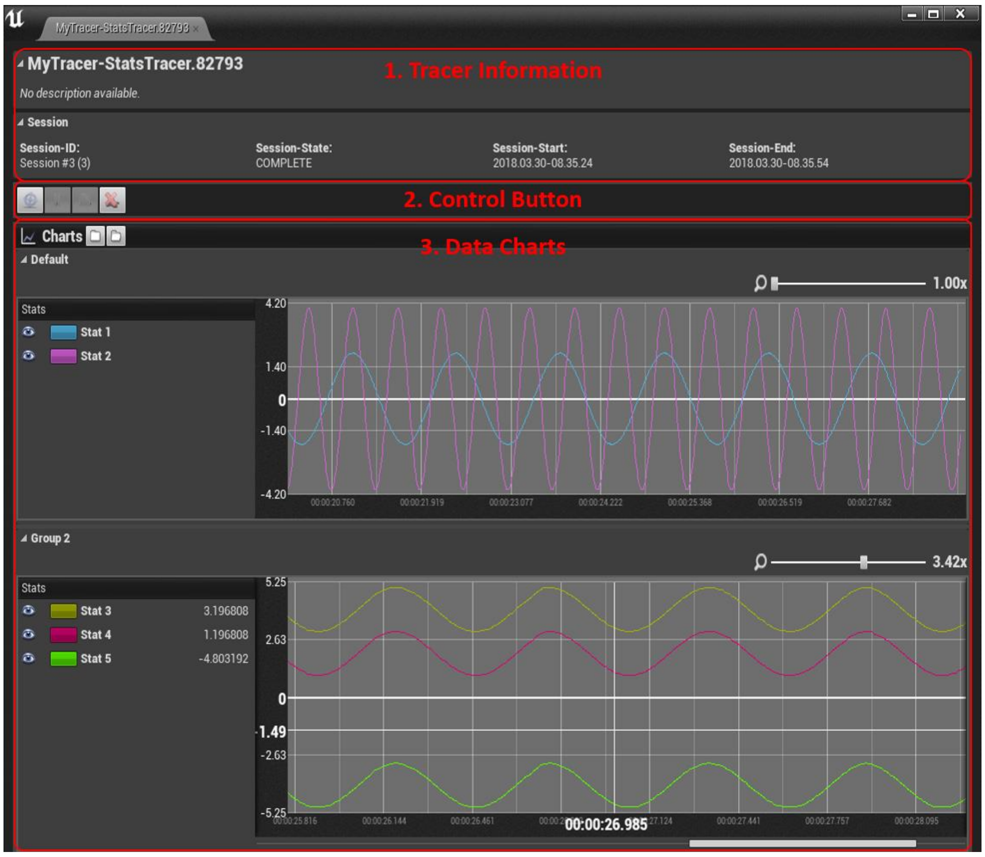
StatsTracer Selection


Tracer Status Icon
Tracer Name and Description
Open Tracer Details Button
Focus Traced Actor Button
Tracer
MyTracer-StatsTracer.83216
MyTracer-StatsTracer.83215
MyTracer-StatsTracer.83213
Pause/Resume Tracer Button
Stop Tracer Button
Delete Tracer Button

Label	Description
Tracer Status Icon	The Tracer Status Icon is indicating the current state of the tracer. 🟢= tracing; ●= paused; 🔴= stopped; 🔵 = complete
Tracer Name and Description	An area showing the tracer's name (top) and its description (bottom).
Open Tracer Details Button	This button opens the tracer's details window. If the details window is already open it will be focused.
Focus Traced Actor Button	Pressing this button will focus the traced actor. When pressing this button during an active session, only the actor in the level hierarchy will be selected. If the using this button when session is completed the actor be additionally focused in scene view. *Note: this button becomes only available (enabled) if the traced actor object instance is still around. Actors that have been traced, but were destroyed cannot be focused.*
Pause/Resume Tracer Button	A toggle button to pause/resume a tracer. Pausing a tracer will stop it from sampling any new data from stats. *Note: this button becomes only available (enabled) if tracer's session is active and the tracer has not been stopped yet.*
Stop Tracer Button	The Stop-Button will stop a tracer from further sampling any data from stats. Once a tracer has been stopped it cannot be reactivated and continue sampling data. *Note: this button becomes only available (enabled) if tracer's session is active and the tracer has not been stopped yet.*
Delete Tracer Button	This button will remove a tracer and all its data from a session freeing allocated memory. *Note: this button becomes only available (enabled) if tracer is in stopped or complete state.*

StatsTracer Details-Window

The tracer's details window shows all its traced stat's sampled data in time-series plots. Stats and their plots are organized in groups. Each group resembles one chart. Charts can be resized, zoomed, collapsed or partially hidden (stats). The details window is structured in three sections. The first section contains to expandable areas. One showing the tracer's name and description and the other the tracer's session information. The second section contains control buttons, which have the same functionality as the ones described in the *StatsTracer Selection* part. The third section is the most interesting one as it contains the data charts for all the traced stats.



MyTracer-StatsTracer.82793

No description available.

Session

Session-ID:	Session-State:	Session-Start:	Session-End:
Session #3 (3)	COMPLETE	2018.03.30-08.35.24	2018.03.30-08.35.54

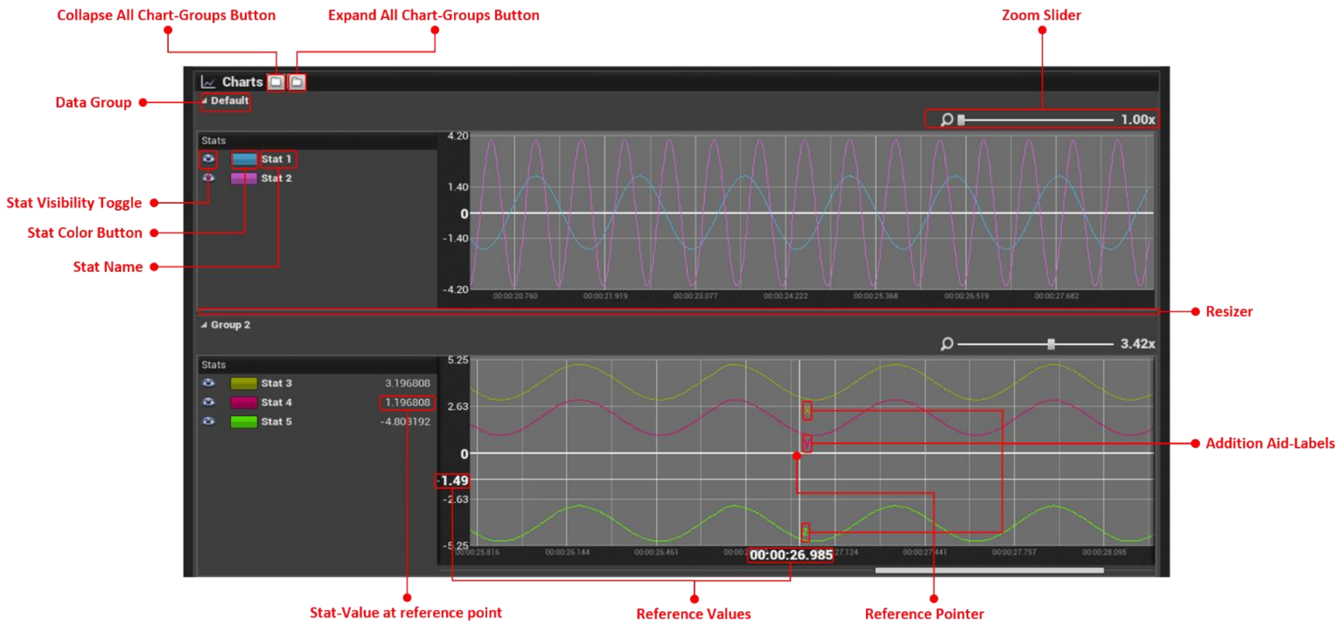
Stats	
Stat 3	3.196808
Stat 4	1.196808
Stat 5	-4.803192

Data Charts

Label	Description
Collapse All Chart-Groups Button	This will collapse all expand (visible) charts.
Expand All Chart-Groups Button	This will expand all collapsed (hidden) charts.
Data Group	A toggle button that shows the name of a data group. Pressing this button will cause the chart area to collapse or expand.
Zoom Slider	A slider control allowing to increase/decrease the zoom of the chart area. Increasing the zoom factor might be useful to detect more subtle changes in plots.
Resizer	The Resizer is a window element that is not visible but is indicated once hovered by mouse. This widget can be used to resize the data chart area in height. To do so move the mouse cursor in the area between to groups or somewhere below the X-Axis. The cursor icon

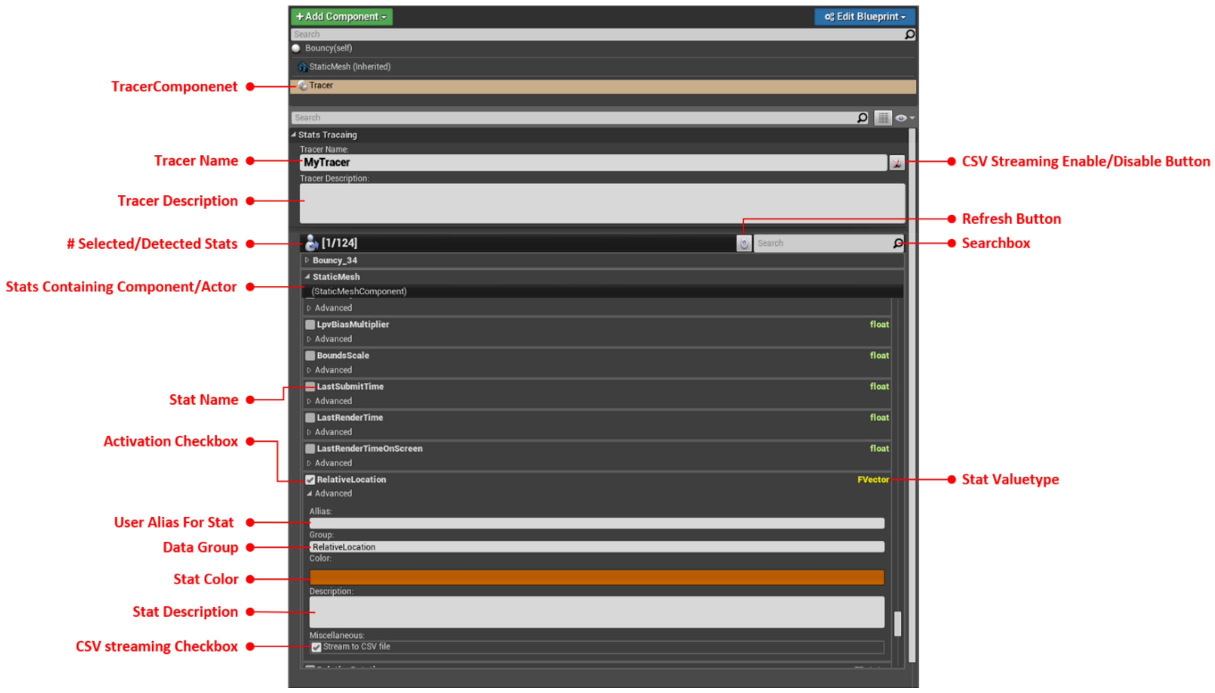
will change to an up-down cursor icon. Press and hold the left mouse button and start dragging to change the height of the area. To stop the resizing just release the left mouse button.

Stat Visibility Toggle	The visibility toggle determines the current visibility state of a stat. If visibility is turned off (the eye icon is not shown) a stat will not be plotted in the chart area. This might be useful if too many stats are plotted at once and their changes overtime cannot be clearly seen in the chart area.
Stat Color Button	The color button can be used to bring up a color picker to change the stat's current color. This button's tooltip also shows the stat's description, if provided.
Stat Name	A label showing the stat's name.
Stat-Value at reference point	A label showing the actual value of the stat at a certain frame/time.
Reference Values	If the option "Show reference pointer" (see Settings section) is enabled two orthogonal reference lines will be shown. One horizontally aligned and another orthogonal/vertical aligned to the mouse cursor. This marks the current point of reference the cursor is pointing to. This point is denoted by a y-Axis value on the left and an timeline/index value on the x-Axis. The index value can be a frame number or time (this can be changed in the settings, see *X-axis timeline mode*).
Additional Aid-Labels	If the option "Display additional labels" (see Settings section) is enabled additional labels for multi-dimensional stats (*FVector*, *FRotator* and *FTransform*) are shown on plots once indexed by mouse cursor. Since multi-dimensional stats elements are plotted with the same color it might become hard to distinguish them. These additional labels can help.



TracerComponent Details-Panel

The TracerComponent details panel will be visible when selecting the Tracer-Component of an actor in the level hierarchy. It displays all the tracer’s properties: name, description and if it is streaming data to csv file. The more important part of this panel is the auto-detected stats overview. When a TracerComponent is first attached to an actor or the refresh button is pressed it will gather information about all the actors and its components properties. All this information will be presented in lists. There will be one list per component (and one for the actor itself). The user is now free to pick from these lists which properties should be traced when starting a new session.



Label	**Description**
TracerComponent	The TracerComponent attached to the actors hierarchy.

Label	**Description**
TracerName	The tracer name. This field is initialized with a default tracer name assembled from actor’s level hierarchy name but can be renamed by user.
TracerDescription	Optionally user provided description for this tracer.
CSV Streaming Enable/Disable Button	Toggle to enable/disable csv streaming for this tracer. *Note: The root directory for csv files to be saved is specified by “Csv file location” option in Settings.*
# Selected/Detected Stats	Information label which shows the currently number of auto-detected stats (right value) and the number of currently selected stats for tracing (left value) *Note: This values update according to specified filter term.*
Refresh Button	Manually detect stats in actor and its attached components. This might be necessary if new properties have been added to components or the actor since the attachment of the TracerComponent.

	Note: Updating manually won’t clear a selection of stats made so far by the user.
Searchbox	A simple quick filter option. The user can enter a filter criteria here which will be used to filter for stats containing the entered criteria in their name. For instance: “Location” will find Relative<u>Location</u> and any other stat with the term “Location” in its name.
Stats Containing Component/Actor	The name of the component or actor containing the stats shown in the list.
Stat Name	The stat name.
Stat Valuetype	The value type of a stat. Possible value type are: bool, byte, int, float, FVector, FRotator and FTransform. Each type is displayed in a different color which is close to the one found in the blueprints editor for variables.
Activation Checkbox	Checkbox representing if a trace should be traced.
User Alias For Stat	A user provided alias for the stat. If an alias is provided this will override the original name of the stat when displayed in data chart.
Data Group	The group in which the stat should be put into. Stats can be grouped with other stats this will cause them to be plotted in the same chart. By default stats will be put into the “Default” group. *Note: Multi-dimensional stat types (FVector, FRotator and FTransform) are put into a group with the same name as the stat itself. This enhances the overall representation when plotting data.*
Stat Description	An optional user provided description.
CSV Streaming Checkbox	CSV streaming can be disabled for single stats by deselecting this checkbox. This way stats can be monitored in data charts but will not be streamed to file.

Settings

General

Tracer databuffer size	Specifies the initial and maximum buffer size for data sources. Tracer will store sampled stats-data into these buffers (one buffer per stat). If the buffer is full the oldest sample is dropped and the new one is inserted. **Min**.: 512, **Max**.: 16384 *Note: Changes of this value will not affect an already running session.*
Tracer sample frequency (Frames)	The frequency how often stats will be sampled. A value of 1 means every frame, 2 every two frames and so on.
Maximum stored session	The upper limit of maximum temporarily stored sessions. If the number of sessions exceed this limit the oldest session is discard. This option can be used to keep memory usage to a minimum.

	Note: Use a zero value if you do not wish a session limitation.
StatsTracer memory cap (Mbyte)	Specifies an upper memory usage limit for the StatsTracer plugin. If the limit is reached no more tracer instances will be created. The user must take action and remove sessions or single tracers to free up memory or increase the limit.

Chart Visual Appearance

Show grid	If enabled a grid will be shown for data charts.
Show reference pointer	If enabled a horizontal and vertical index line will be rendered under the mouse cursor. In addition, the current index frame number and Y-Axis value is displayed.
Display additional labels	If enabled additional text labels will be displayed for multi-dimensional values (FVector, FRotator and FTransform).
X-axis grid scale (Pixel)	Changes the grid size along the X-Axis.
Y-Axis grid scale (Pixel)	Changes the grid size along the Y-Axis.
X-axis timeline mode	This option changes the way labels are displayed on the X-axis. There are five modes • None – Nothing is shown on the X-Axis • Time – The elapsed game time is shown (default) • Relative Time – The relative time to the last sampled data point is shown • Frame – Elapsed game frames are shown • Relative Frame – The relative frames to the last sampled data point are shown
Reference pointer timeline mode	Changes the label shown by the reference pointer. Same options as for the X-Axis.

Csv Setting

Csv file location	The root directory for streamed csv data.

Tracer Component

Global stats filter	A list of strings used as filters by the “auto detected” functionality of the TracerComponent. If stats names match any of these strings they won’t be shown in the TracerCompoment. *Note: You can use regular expressions here.*

HowTo

In this section you will learn how to setup a StatsTracer. To do so we can either use the TracerComponent, the blueprint functions or a combination of both.

Before starting to create a StatsTracer we first need a target. So, let's make up something simple. Imagine our game contains a custom blueprint actor: *TestObject*. When watching the *TestObject* in play-in-editor mode (PIE), we can observe the following behaviour:

- it is moving up and down,
- expanding and shrinking,
- rotating around the Z-Axis and
- changing color.

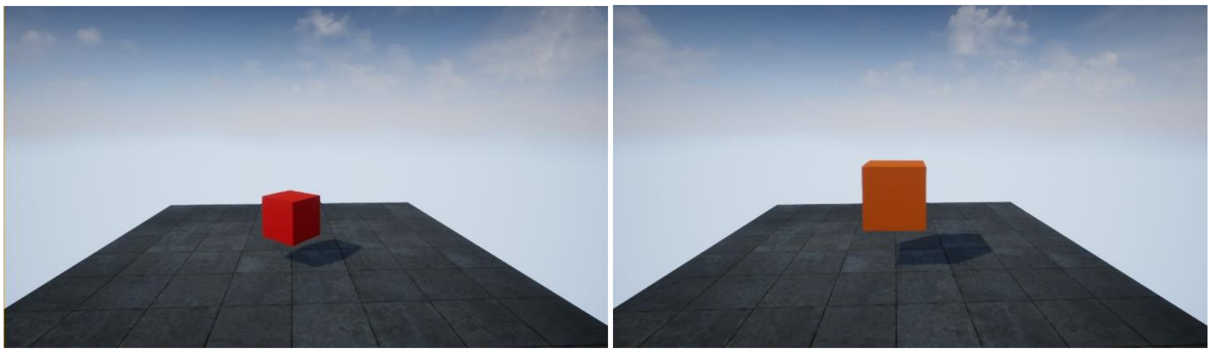


Figure 5: Dancing cube example.

Figure 5 depicts two snapshots at different points in time of the observed *TestObject*. If you are keen to watch a live presentation you can check out this YouTube clip: https://www.youtube.com/watch?v=mE0ZQu1p9aA.

Ok, now let's say we are interested in a detailed view of how the actor's transformation and color values are changing over time. This is the moment you want to create a StatsTracer. First let's start with the blueprint version.

Blueprint Functions

Creating a new StatsTracer in blueprints is rather simple. Just open up the context-menu and start typing "*Create Stats Tracer*". If you cannot find any functions try to disable the "*Context Sensitive*" option in the top-right corner. You should be able to find the blueprint functions shown in Figure 6. Now, to setup a new StatsTracer we will continue with the following instructions:

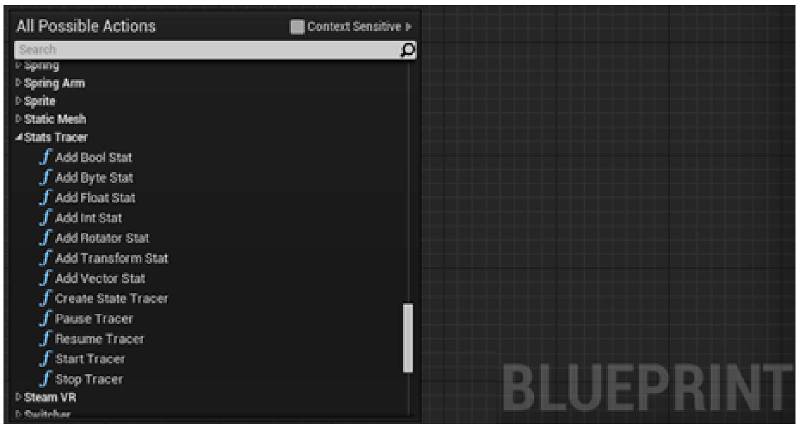
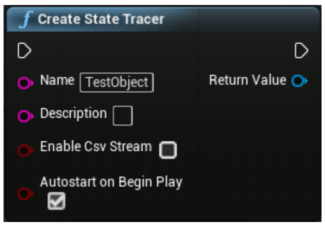
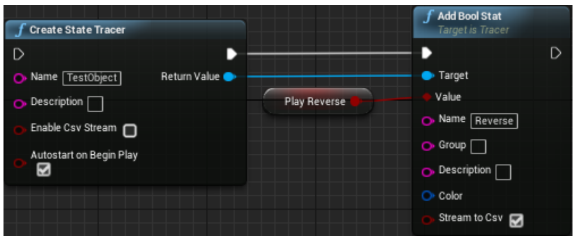


Figure 6: StatsTracer blueprint functions.

1. First, we create a "*Create Stats Tracer*" blueprint node
    a. We provide a name: "*TestObject*",
    b. Some description so we know what's that tracer is doing
    c. Optionally we could enable the "*Enable Csv Stream*" functionality so all the sampled data will be streamed into a csv file



2. Next, we choose variables from the blueprint panel and drag them out as getter's
    a. In this example there are three custom variables we want to trace: '*Play Reserve*' (bool), '*t*' (float) and '*Color*' (FVector)
    b. To get access to the transformation values we use the *RootComponent*
3. From the "*Create Stats Tracer*" node we drag-out a new connection and type in "*Add* ". With the "*Context Sensitive*" option enabled we should see functions like "*Add Bool Stat*", "*Add Float Stat*" and so on. To create a new data-source for the '*Play Reserve*' variable we use the "*Add Bool Stat*" blueprint function.
    a. We plug in the 'Play Reserve' variable into the '*Value*' slot
    b. Enter a name for the variable's data-source, which will be visible in data-charts later
    c. Optionally we could put this data-source into an arbitrary group by simply specifying a name in '*Group*'
        i. If left empty data-sources are put into the 'Default' group, except multi-dimensional types like a vector. They will be put into a group with the same name as specified in b)
    d. We could provide a description and color or disable csv streaming of this data-source



4. We will repeat step three for '*t*', '*Color*', '*RootComponent.RelativeLocation*', '*RootComponent.RelativeRotation*' and '*RootComponent.RelativeScale3D*' with the appropriate "*Add*" methods.
5. Finally, we will create a "*Start Tracer*" node and add it to the end of this execution chain.

Our final result should look something like the blueprint shown in Figure 7. A good place for the execution of this code is the "*BeginPlay*" event, but of course you can execute this where ever you want.

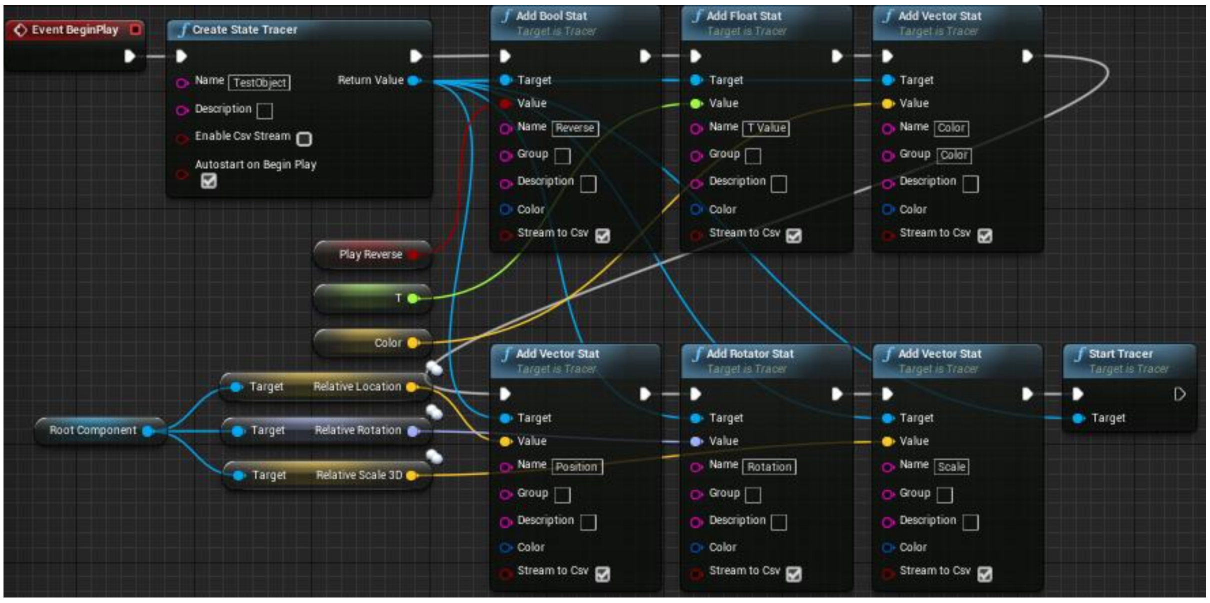
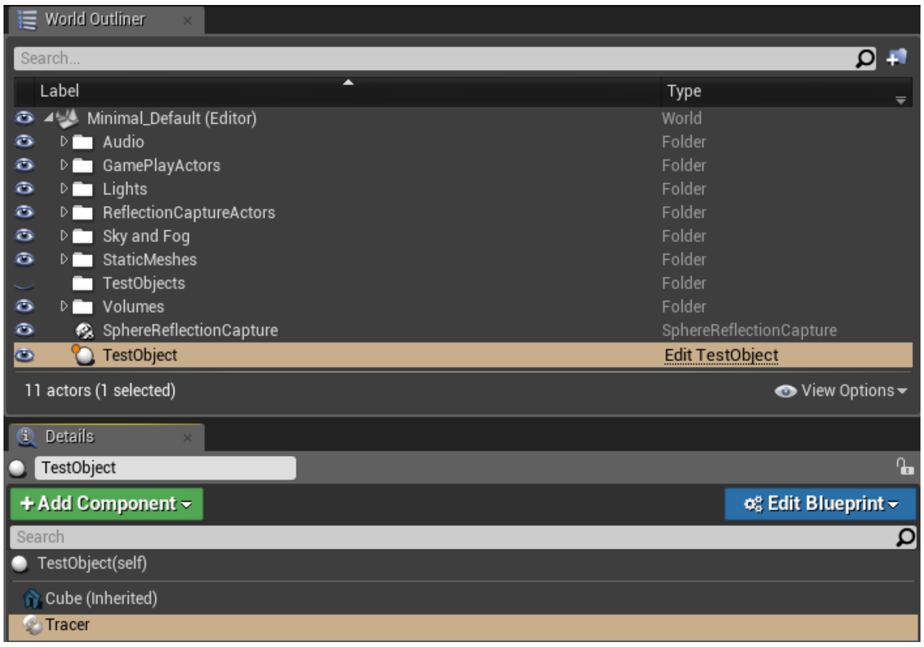


Figure 7: Creating a new StatsTracer in blueprint.

TracerComponent

Alternatively, to blueprints we can achieve the same goal by using the TracerComponent. In the "*World Outliner*" window we select the *TestObject* and use the "*Add Component*" button to add the TracerComponent. Simple search for "*Tracer*" and hit enter.


World Outliner

Search...

Label	Type
Minimal_Default (Editor)	World
Audio	Folder
GamePlayActors	Folder
Lights	Folder
ReflectionCaptureActors	Folder
Sky and Fog	Folder
StaticMeshes	Folder
TestObjects	Folder
Volumes	Folder
SphereReflectionCapture	SphereReflectionCapture
TestObject	Edit TestObject

11 actors (1 selected) View Options

Details

TestObject

+ Add Component | Edit Blueprint

Search

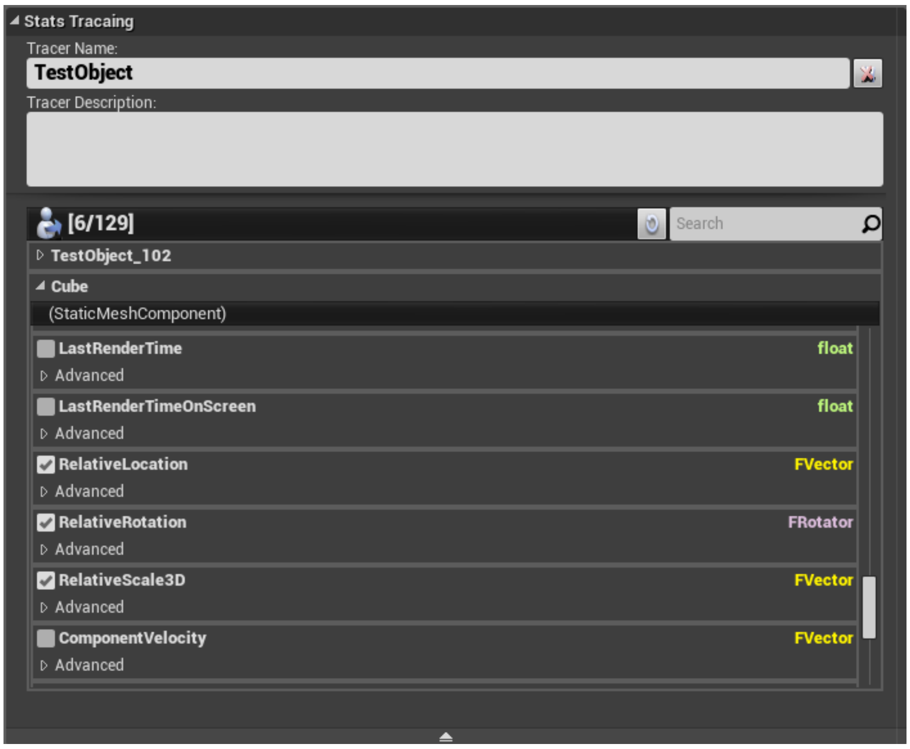
TestObject(self)

Cube (Inherited)

Tracer

Figure 8: Adding a TracerComponent to the TestObject.

After adding the TracerComponent to the TestObject, go to the “*Details*” panel and select it. You will see the following view:


Stats Tracing

Tracer Name:

TestObject

Tracer Description:

[6/129]

Search

TestObject_102

Cube

(StaticMeshComponent)

☐ LastRenderTime	float
Advanced	
☐ LastRenderTimeOnScreen	float
Advanced	
✓ RelativeLocation	FVector
Advanced	
✓ RelativeRotation	FRotator
Advanced	
✓ RelativeScale3D	FVector
Advanced	
☐ ComponentVelocity	FVector
Advanced	

Figure 9: Creating a StatsTracer by adding a TracerComponent.

The TracerComponents details-panel will provide us with a nice list of all tracible properties belonging to the *TestObject*. We simply pick the ones we are interested in. For a more information about the TracerComponent details-panel see TracerComponent Details-Panel section.

The final tracer output looks the same for both versions. The ‘*T Value*’ and ‘*Reserve*’ data-source are put into the ‘*Default*’ group and therefore sharing the same data-chart. *Color*, *Location*, *Rotation* and *Scale3D* are put into separate groups and enjoying their own data-charts.

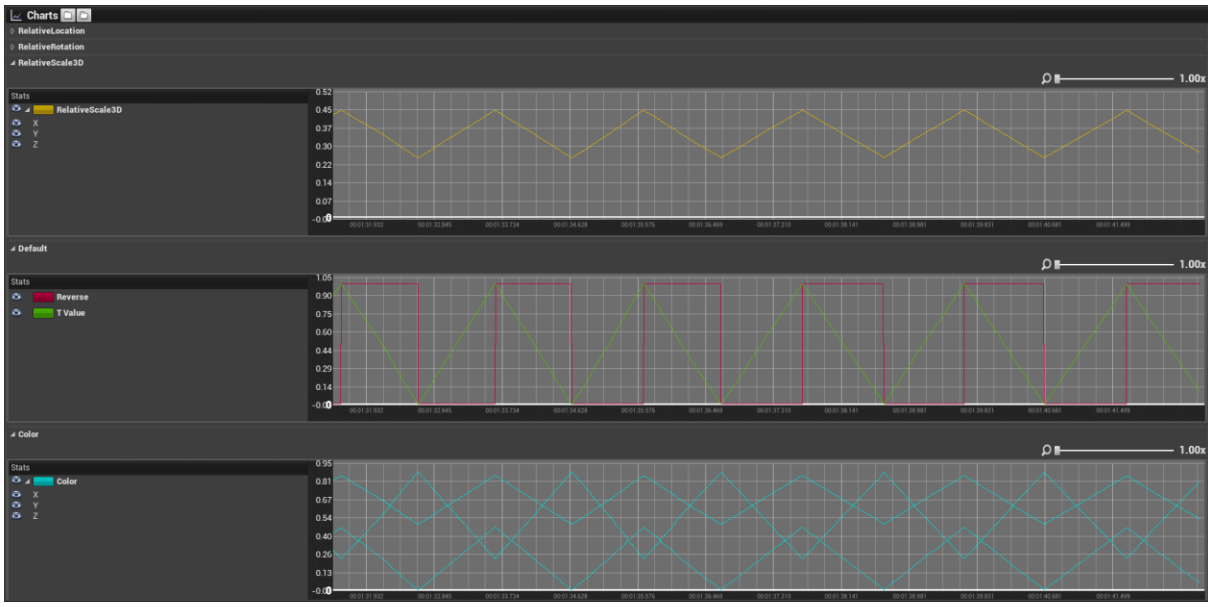


Figure 10: StatsTracer example output.

Bonus: Creating StatsTracer in C++

Using blueprint functions or the TracerComponent for tracing stats only works on UPROPERTY exposed variables. If you find yourself in a situation where you cannot rely on UPROPERTY but have access to the class which has certain stats you want to monitor, you can actually do so in C++ code.

First you need to modify your projects *.uproject* file and add "*StatsTracer*" as addition dependency.

```
…
"Modules":
[
  {
    "Name": "MyProject",
    "Type": "Runtime",
    "LoadingPhase": "Default",
    "AdditionalDependencies":
    [
       "Engine",
       "StatsTracer" // ← needs to be added!
    ]
  }
],
…
```

Now you have access to the *Tracer.h* header file, that contains all the functionality you need to create a tracer. Let's see an example.

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "MyActor.generated.h"

UCLASS()
class STATSTRACERTESTCPP_API AMyActor : public AActor
{
	GENERATED_BODY()

	// Our private variable we are interested in
	bool MyBool;

public:
	// Sets default values for this actor's properties
	AMyActor();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;
};
```

Table 1: MyActor.h

```cpp
#include "MyActor.h"
#include "Tracer.h" // Provides access to UTracer class

AMyActor::AMyActor()
{
        PrimaryActorTick.bCanEverTick = true;
}

void AMyActor::BeginPlay()
{
        Super::BeginPlay();

        // Create a new tracer object instance
        UTracer* tracer = NewObject<UTracer>();
        {
                // Initialize the tracer
                tracer->Initialize("TestObject", "", this);

                // Add our stat here
                tracer->AddBoolStat(&(this->MyBool), "MyBool");
                // Add more stats ...

                // This is actually not required as all tracer created
                // at 'BeginPlay' will be automatically started,
                // if 'auto-start' flag is set (which is true by default)
                tracer->StartTracer();
        }
}

void AMyActor::Tick(float DeltaTime)
{
        Super::Tick(DeltaTime);

        // Change our stat
        this->MyBool = FMath::RandBool();
}
```
Table 2: MyTracer.cpp

Table 1 shows the declaration of a custom actor class *MyActor*. *MyActor* has a private member variable which is not exposed to the editor by a UPROPERTY tag and therefore we cannot use blueprints or the TracerComponent to access it. But since we are the owner of this class we can create a tracer directly from the C++'s end. In Table 2 we can see how to do this. First, we include the *Tracer.h* header to get access to the *UTracer* class. In the *BeginPlay* method we create a new tracer similar to the way we did in blueprints. We use the *NewObject* utility method to create a new instance of a tracer object. The tracer object provides the same function set as you have already seen in blueprints plus one additional method. **Before adding stats, we must call the *Initialize* method with a name, description and a reference to *this* object instance.** After the initialization is done our newly created tracer will be registered in the active tracer session and we are now ready to add stats.

Note: The UTracer is a proxy class. That is, it is merely used to get access to a data-repository for the target object. This will happen when calling the Initialize method. If two different UTracer object instances invoke their Initialize method with the same target (this) object, both will have access to the same data-repository.

Blueprint vs. TracerComponent vs. C++ Code

You might wonder why there are three different ways to create StatsTracer and which way is the one to go. The answer for this is: it's your choice. But if you have read the previous sections you probably got an idea about the pros and cons for each method. The following table will summarize them for you.

TracerComponent

- ✓ The easiest way to create a StatsTracer
- ✓ All accessible stats are nicely presented in lists grouped by components and actor
- ✓ Adding/Removing stats or changing their settings is on-the-fly
- ✗ Only available for objects in the "*World Outliner*" window
- ✗ Only effects the object instance the TracerComponent was attached to
- ✗ No access to non UPROPERTY variables

Blueprints

- ✓ More flexible way to create a StatsTracer
- ✓ Ability to control a tracer (pause/resume, stop)
- ✓ Each dynamically created blueprint object will have its own tracer object
- ✗ Changes are not on-the-fly and require more effort
- ✗ No access to non UPROPERTY variables

C++ Code

- ✓ Most flexible way to create a StatsTracer
- ✓ Allows access to non UPROPERTY variables
- ✗ Not very user-friendly usage
- ✗ Changes are not on-the-fly and require more effort

Conclusion

As a conclusion of all this you will most likely use the TracerComponent in cases where you would like to trace more individual objects, e.g. an enemy boss creature with unique behaviours. In a scenario where you want to trace stats of multiple objects of the same kind, e.g. a generator-building; each holding a unique state of current voltage etc., you will surly want to use the blueprint functions or C++ code (if possible). C++ code however should only be used as a last resort. As for instance in cases where variables are not exposed by UPROPERTY. Otherwise you should stick to the blueprints.

Disclaimer