Low-Level Memory Tracker

Track memory usage in your Unreal Projects.

The **Low-Level Memory Tracker (LLM)** is a tool that tracks memory usage in **Unreal Engine (UE)** projects. LLM uses a scoped-tag system to keep an account of all memory allocated by the Unreal Engine and the OS. LLM supports all platforms used by Unreal Engine.

LLM Trackers

There are currently two trackers in LLM. Each tracker has its own allocation map and tag stack. The Default Tracker is for all allocations from the Engine. It is the higher-level of the two, and records allocations made through the `FMemory` class function, `Malloc`. This is the tracker that supplies the stats for the `stat LLM` and `stat LLMFULL` Console Commands. The Platform Tracker is the lower-level version, and it records all allocations made from the OS. For example, it will track the internal allocations made by functions like `Binned2`. The Default Tracker stats are therefore a subset of the platform tracker stats.

LLM Setup

To enable LLM in your project, use the following command-line arguments and Console Commands.

Command Line Argument	Description
`-LLM`	Enables LLM
`-LLMCSV`	Continuously writes out all values to a CSV file. Automatically enables -LLM.
`-llmtagsets=Assets`	Experimental Feature. Shows the total allocated by each asset.
`-llmtagsets=AssetClasses`	Experimental Feature. Shows the totals for each UObject class type.

Console Commands	Description
`stat LLM`	Shows the LLM summary. All lower level engine stats are grouped under a single Engine stat.
`stat LLMFULL`	Shows all LLM stats.
`stat LLMPlatform`	Shows stats for all memory allocated from the OS.
`stat LLMOverhead`	Shows the memory used internally by LLM.

When using the `-LLMCSV` command-line argument, the `.CSV` file will be written out to `saved/profiling/llm/`. The file will contain a column for each tag showing the current value in MB. A new line is written every 5 seconds (by default). The frequency can be changed using the `LLM.LLMWriteInterval` Console Variable.

LLM Tags

Every memory allocation made by the Engine (including game code) is assigned a tag value identifying the category to which it belongs. This means that all memory is tracked once and

only once, nothing is missed, and nothing is counted twice. The total of all of the categories will add up to the total amount of memory used for the game.

The tags are applied using tag-scope macros. Any allocation made within that scope will be given the specified tag. LLM maintains a stack of tag scopes and applies the top tag to the allocation. LLM stats can be viewed in-game using the `stat LLM` or `stat LLMFULL` Console Commands. The current total for each tag will be shown in MB. LLM also writes out the stat values to a `.CSV` file, so that the values can be analyzed. The following Tag Categories currently exist within the Engine:

Tag Name	Description
UObject	This includes any class that inherits from `UObject` and anything that is serialized by that class including properties. **UObject** is a catch-all for all Engine and game memory that is not tracked in any other category. Note that this stat doesn't include Mesh or Animation data which are tracked separately. It corresponds to the number of Objects placed in the Level.
EngineMisc	Any low-level memory that is not tracked in any other category.
TaskGraphTasksMisc	Any task that is kicked off from the task graph that doesn't have its own category. This should generally be fairly low.
StaticMesh	This is the `UStaticMesh` class and related properties, and does not include the actual mesh data.

Custom Tags

When profiling memory usage for your project with Unreal Insights, you may discover that there is allocated memory tagged as LLM Untracked in Memory Insights. The `LLM_DECLARE_TAG` and `LLM_DEFINE_TAG` macros are used to create custom tags to help you discover untracked memory allocations. These macros do not require modifying the engine files and can be done in game modules or plugins. When you want to use the custom tag you created, use the `LLM_SCOPE_BYTAG` macro. The steps are summarized as follows:

1. Declare a custom LLM tag in your header file with `LLM_DECLARE_TAG`.

2. Define your custom LLM tag in the associated `.cpp` file with `LLM_DEFINE_TAG`.
3. Use the `LLM_SCOPE_BYTAG` where you wish to track memory usage in your `.cpp` file with `LLM_SCOPE_BYTAG`.

Refer to the Example below for a small example declaring and using a custom tag.

Custom Tag Macros

LLM_DECLARE_TAG

The `LLM_DECLARE_TAG` macro declares a tag which is defined elsewhere that can be used in `LLM_SCOPE_BYTAG` or referenced by name in other `LLM_SCOPE` s.

- **Parameters**
  - `UniqueName`: The name of the tag for looking up by name. Must be unique across all tags passed to `LLM_DEFINE_TAG`, `LLM_SCOPE`, or `ELLMTag`.

LLM_DEFINE_TAG

The `LLM_DEFINE_TAG` macro defines a tag which can be used in `LLM_SCOPE_BYTAG` or referenced by name in other `LLM_SCOPE` s.

- **Parameters**
  - `UniqueNameWithUnderscores`: Modified version of the name of the tag. Used for looking up by name. Must be unique across all tags passed to `LLM_DEFINE_TAG`, `LLM_SCOPE`, or `ELLMTag`. The usual separator for parents (`/`) must be replaced with (`_`) in `LLM_DEFINE_TAG`s.
  - `DisplayName`: (Optional) The name to display when tracing the tag joined with (`/`) to the name of its parent if it has a parent or `NAME_None` to use the `UniqueName`.
  - `ParentTagName`: (Optional) The unique name of the parent tag or `NAME_None` if it has no parent.
  - `StatName`: (Optional) The name of the stat to populate with this tag's amount when publishing LLM data each frame or `NAME_None` if no stat should be populated.
  - `SummaryStatName`: (Optional) The name of the stat group to add on this tag's amount when publishing LLM data each frame or `NAME_None` if no stat group should be added to.

Example

CustomTagExample.h

```
#pragma once
...
LLM_DECLARE_TAG(MyTestTag);

```

Copy full snippet

CustomTagExample.cpp

```
LLM_DEFINE_TAG(MyTestTag);

AMyActor::AMyActor()
{
LLM_SCOPE_BYTAG(MyTestTag);
MyLargeBuffer.Reset(new uint8[1024*1024*1024]);
}
```

Copy full snippet

Tag Sets (Experimental)

To use Tag Sets, define `LLM_ALLOW_ASSETS_TAGS` in `LowLevelMemTracker.h`. When using Tag Sets, each allocation will additionally store the Asset name or the Object class name.

> Using Tag Sets will add overhead to both memory usage and runtime performance.

Technical Implementation Details

LLM works by maintaining a map of all allocations indexed by a pointer. The map currently contains the size of each allocation and its assigned Tag. Games can have as many as 4

million live allocations at any one time, so it is important to keep the memory overhead as small as possible. The current implementation uses 21 bytes per allocation:

Allocation	Size
Pointer	8 bytes
Pointer Hash Key	4 bytes
Size	4 bytes
Tag	1 byte
Hash Map Index	4 bytes

When an allocation is tracked with the `OnLowLevelAlloc` function, the tag on the top of the tag stack becomes the current tag and is stored in the allocation map with that pointer as its key. To avoid contention, the frame deltas of each tag are tracked in a separate `FLLMThreadState` class instance. At the end of a frame, these deltas are summed and published to the stats system and `.CSV` file.

LLM is initialized very early on, which means that it must be enabled by default. If LLM isn't enabled on the command line, it will shut itself down and clean up all memory, ensuring there is no overhead. LLM is completely compiled out in Test and Shipping builds.

LLM can be run without the stats system (for example in the Test config). It won't be possible to show the stats on screen, but the stats will still be written out to the `.CSV` file. LLM will need to be enabled by modifying `ENABLE_LOW_LEVEL_MEM_TRACKER` in `LowLevelMemTracker.h`.

Tags are applied using the scope macros. The two main macros are:

- `LLM_SCOPE(Tag)`
- `LLM_PLATFORM_SCOPE(Tag)`

These set the current scope of the Default Tracker and Platform Tracker, respectively. There are platform-dependent versions of these scopes, for example, `LLM_SCOPE_[Console](Tag)`, which use the platform-specific tag enumerated types. The scope macros that use stats,

such as `LLM_SCOPED_TAG_WITH_STAT`, are considered deprecated at the moment, and should not be used.

All memory used by LLM internally is managed by the platform-supplied `LLMAlloc` and `LLMFree` functions. It is important that LLM doesn't make allocations in any other way, so that it doesn't track its own memory usage (and cause infinite recursion).

Additional Technical Details

The following section contains various notes and additional information you should be aware of when using LLM.

- Because the overhead of LLM can be 100MB or more it is strongly advised to run in large memory mode on consoles.
- LLM in the Test config won't show the on-screen stats page, but it will write out the `.CSV` file. LLM is completely disabled in Shipping.
- Asset tag tracking is still in an early experimental state.