



Table of Contents

Introduction.....	2
Target audience.....	2
Experimental warning.....	2
UnityEngine.MemoryProfiling API does not work in some Unity versions.....	2
Contact.....	3
Can I use this tool even when I work on a commercial project?.....	3
Heap Explorer <> Memory Profiler Comparison.....	4
How to install Heap Explorer.....	5
How to capture a memory snapshot.....	6
Brief Overview.....	8
Compare Memory Snapshots.....	9
C# Objects.....	10
C# Object Inspector.....	12
References / Referenced by.....	16
Root Path.....	17
C# Object Duplicates.....	19
C# Delegates.....	20
C# Delegate Targets.....	22
C# Static Fields.....	23
C# Memory Sections.....	25
C++ Objects.....	26
Exclude NativeObject connections.....	29
C++ Asset Duplicates (guessed).....	30
Observations.....	32
Known Unity Issues.....	34
Feedback & Questions on Unity Memory Profiling API.....	39
Changelog.....	40

Introduction

Heap Explorer is a memory profiler, debugger and analyzer for Unity.

I spent quite some time identifying and fixing memory leaks, as well as looking for memory optimization opportunities in Unity applications in the past. During this time, I often used [Unity's MemoryProfiler](#) and while it's a useful tool, I never was entirely happy with it.

I then decided to write my own memory profiler, that helps me in future projects to achieve my work more easily and faster.

Target audience

Heap Explorer is a tool for programmers and people with a strong technical background, who are looking for a tool that helps them identifying memory issues and memory optimization opportunities in Unity applications.

Heap Explorer is not fixing memory leaks, nor optimizing content for you automatically. Heap Explorer is a tool whose data you have to interpret and draw your own conclusions from.

Experimental warning

Heap Explorer uses Unity's experimental MemoryProfiling API, which contains [various bugs](#), from cosmetics to major issues that make memory snapshots not trustworthy. These bugs occur in every application that make use Unity's MemoryProfiling API, such as Unity's own MemoryProfiler tool and are not limited to Heap Explorer. I reported all bugs I found and hope Unity Technologies is able to fix them.

UnityEngine.MemoryProfiling API does not work in some Unity versions

Unity 2017.3 and earlier are not supported by Heap Explorer on purpose.

The MemoryProfiling API does not work in Unity 2017.4-2018.2 with Scripting Runtime .NET 4.x.

Unity 2017.4.6f1 and later 2017.4 releases work with Scripting Runtime .NET 3.5.

Unity 2018.2.6f1 works with Scripting Runtime .NET 3.5.

Please see the Known Issues section in this document to get an idea of the existing Memory Profiling API issues I found (there are probably more).

Contact

You can get in touch with me via the Unity forums or email.

Post in the Heap Explorer thread:

<https://forum.unity.com/threads/wip-heap-explorer-memory-profiler-debugger-and-analyzer-for-unity.527949/>

Use the “Start a Conversation” functionality to contact me via the forum:

<https://forum.unity.com/members/peter77.308146/>

Can I use this tool even when I work on a commercial project?

Yes, you can use Heap Explorer to debug, profile and analyze your hobby-, indie- and commercial applications for free. You do not have to pay me anything.

If Heap Explorer helped you, I would definitely appreciate a mentioning in your credits screen. Something like “Heap Explorer by Peter Schraut” would be very much appreciated from my side. Again, this is not required at all.

You can use Heap Explorer for free, without having to give me credit or mention you used the tool at all.

Heap Explorer <> Memory Profiler Comparison

The following table shows which features or functionality is available or missing in either tool. The table lists, what I believe are, the most important differences between the two tools.

Both tools provide functionality to capture a memory snapshot and inspect all it's data, but there are a few differences that I would like to point out.

Feature	Heap Explorer	Memory Profiler
Load/save Snapshot as JSON	✗	✓
Load/Save Snapshot as binary	✓	✗
Compare Snapshots	✓	✗
Search and Sort (by name, type, etc)	✓	✗
List Memory Sections	✓	✗
List C# Object Duplicates	✓	✗
Inspect value and reference type objects	✓	✓
Inspect 2D and 3D (jagged) arrays	✓	✗
Inspect value types in arrays	✓	✗
Show all Paths to Root of a C# object	✓	✗

There are other features in Heap Explorer that don't exist in Memory Profiler, but I don't think it's necessary to list them all in the table. Please read through this documentation to get an idea what Heap Explorer can do for your instead.

How to install Heap Explorer

Download the latest version from:

<http://www.console-dev.de/bin/heapexplorer.zip>

Once downloaded, extract to the zip archive to any location under “Assets/” in your Unity project.

I recommend to extract the zip archive to “Assets/Editor/”. This will install Heap Explorer to “Assets/Editor/HeapExplorer”

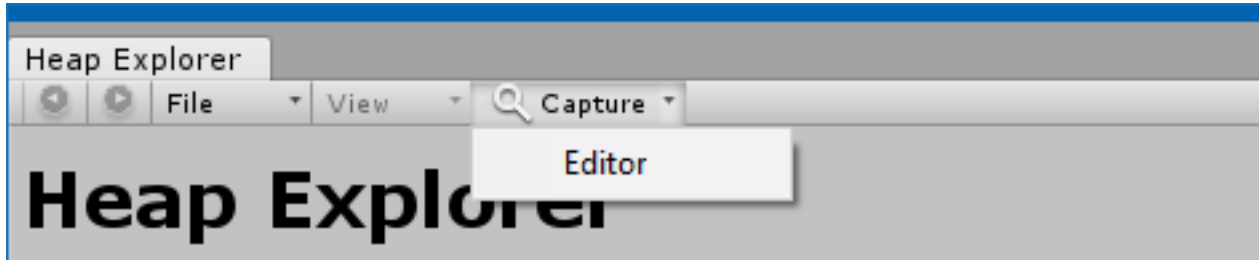
Once Heap Explorer is extracted/copied to this location, Unity will recompile the project and afterwards you can open it from Unity’s “Window > Heap Explorer” menu item.

In order to uninstall/remove the Heap Explorer, simply delete the “Assets/Editor/HeapExplorer” directory.

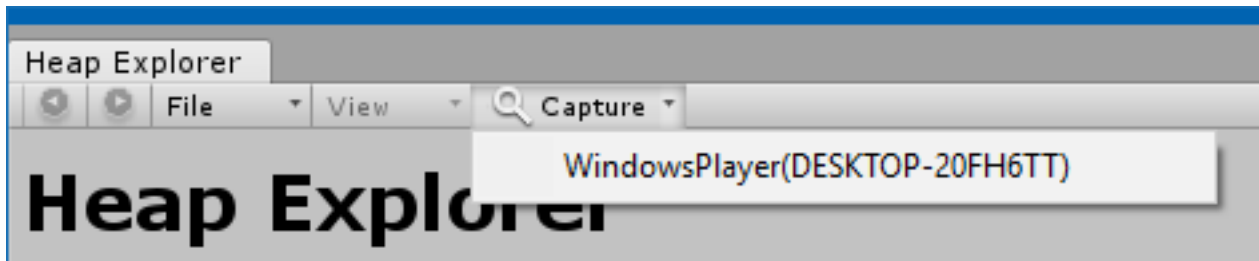
How to capture a memory snapshot

Heap Explorer displays the connected Player in the “Capture” drop-down, which you can find in the toolbar. The button is located under a drop-down menu, to avoid clicking it by accident.

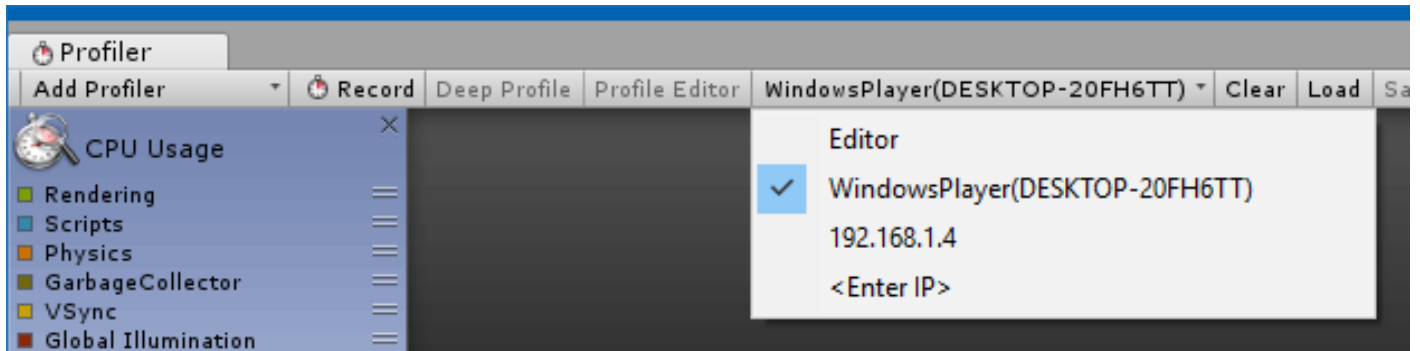
If no Player is connected, Heap Explorer displays “Editor”. Clicking the “Editor” button then captures a memory snapshot of the Unity editor.



If a Player is connected, Heap Explorer displays the Player name, rather than “Editor”. It’s the same name that appears in Unity’s Profiler window.



In order to connect to a certain target, you have to use Unity’s Profiler, as shown below. As you select a different target (Editor, WindowsPlayer, ...) in Unity’s Profiler window, Heap Explorer will update its entry in the “Capture” drop-down accordingly, depending on what is selected in Unity’s Profiler.



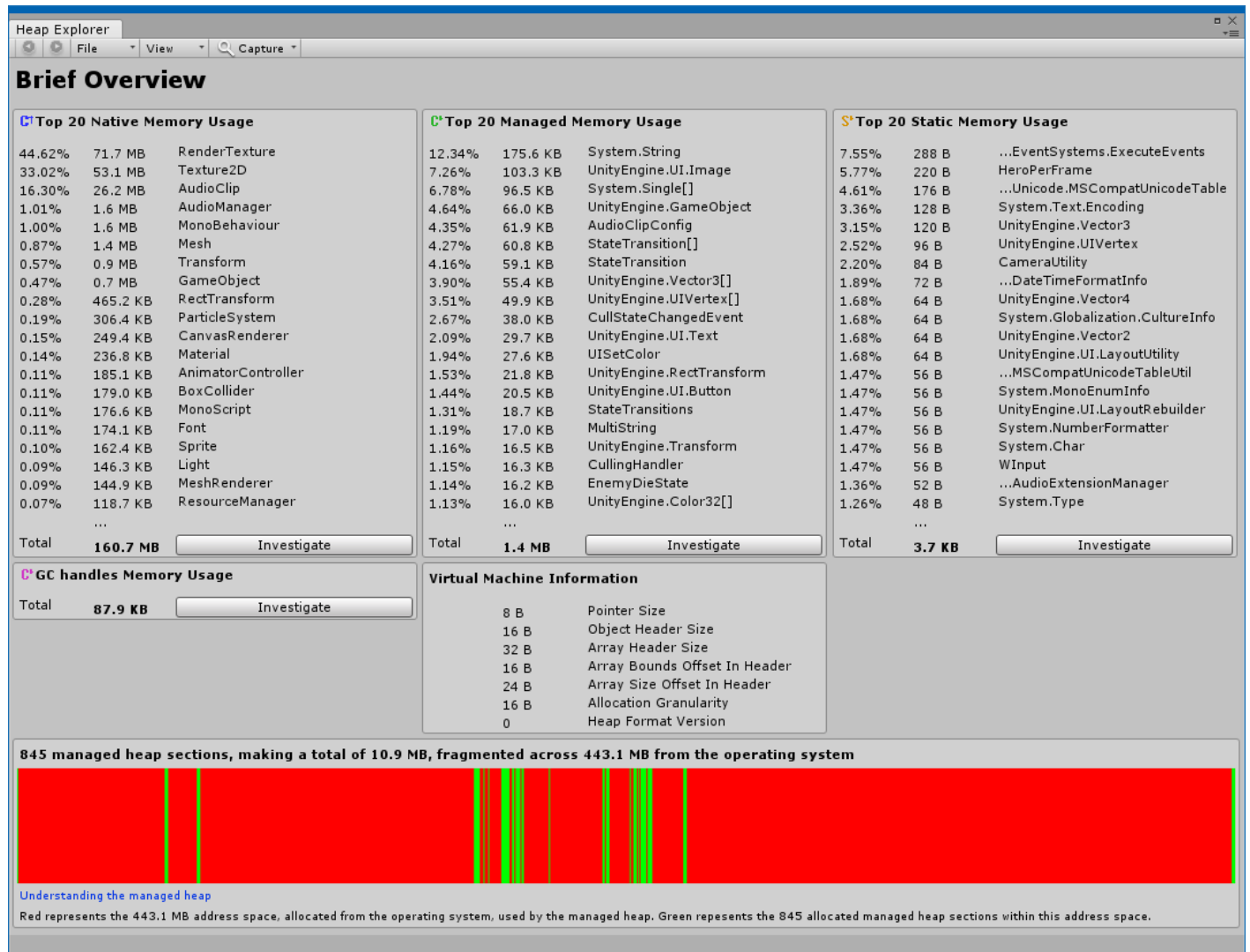
Why can't I select the profiling target in Heap Explorer directly?

The Unity Editor API does not provide public functionality to implement this. It does expose some of the needed functionality in an unsupported namespace only. I don't want to use unsupported and internal API's, because I know from past experience, this is going to cause trouble in the long run, such as breaking the tool with several Unity updates.

Brief Overview

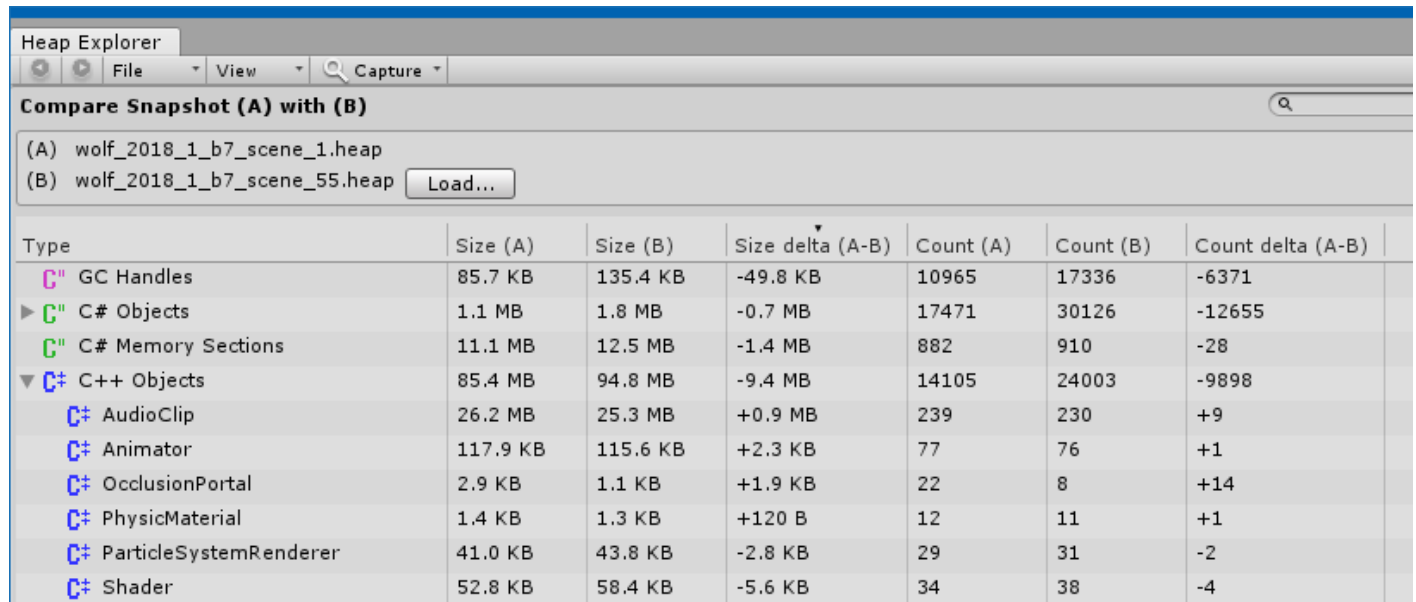
The Brief Overview page shows you various important facts of the memory snapshot in a compact fashion.

This includes the memory used by the top 20 native and managed objects, as well as the fragmentation of the managed heap in the operating system memory.



Compare Memory Snapshots

Heap Explorer supports to compare two memory snapshots and lists the difference between those. This is a useful tool to find memory leaks.



Type	Size (A)	Size (B)	Size delta (A-B)	Count (A)	Count (B)	Count delta (A-B)
GC Handles	85.7 KB	135.4 KB	-49.8 KB	10965	17336	-6371
C# Objects	1.1 MB	1.8 MB	-0.7 MB	17471	30126	-12655
C# Memory Sections	11.1 MB	12.5 MB	-1.4 MB	882	910	-28
C++ Objects	85.4 MB	94.8 MB	-9.4 MB	14105	24003	-9898
AudioClip	26.2 MB	25.3 MB	+0.9 MB	239	230	+9
Animator	117.9 KB	115.6 KB	+2.3 KB	77	76	+1
OcclusionPortal	2.9 KB	1.1 KB	+1.9 KB	22	8	+14
PhysicMaterial	1.4 KB	1.3 KB	+120 B	12	11	+1
ParticleSystemRenderer	41.0 KB	43.8 KB	-2.8 KB	29	31	-2
Shader	52.8 KB	58.4 KB	-5.6 KB	34	38	-4

“A” and “B” represent two different memory snapshots. The “delta” columns indicate changes. The “C# Objects” and “C++ Objects” nodes can be expanded to see which objects specifically cause the difference.

Snapshot “A” is always the one you loaded using “File > Open Snapshot” or captured. While “B” is the memory snapshot that is used for comparison and can be replaced using the “Load...” button in the Compare Snapshot view.

C# Objects

The C# Objects view displays managed objects found in a memory snapshot. Object instances are grouped by type. Grouping object instances by type allows to see how much memory a certain managed object type is using.

The view contains the main list (top-left panel) that contains all managed objects, an [inspector](#) (top-right panel) that displays the fields and field-values of the selected object, one or multiple [paths to root](#) (bottom-right panel) of the selected object as well as what objects the selected object [references and is referenced by](#) (bottom-left panel).

The screenshot displays the Visual Studio Heap Explorer interface. The main list on the left shows various objects grouped by type. The right panel shows the inspector for the selected 'AwardList' object, displaying its fields and values. The bottom-left panel shows references to the selected object, and the bottom-right panel shows the path to the root of the object.

C# Type	C++ Name	Size	Count	Address	Assembly
Agent[]		2.1 KB	3		Assembly-CSharp
AgentId		216 B	9		Assembly-CSharp
AgentId[]		272 B	7		Assembly-CSharp
Array<AbstractWeapon.Frame>		160 B	5		Assembly-CSharp
Array<HUDMarker>		32 B		0x2799F36C480	Assembly-CSharp
Array<StateHandler.Entry>		160 B	5		Assembly-CSharp
Array<UnityEngine.ParticleSystem.Particle>		32 B		0x2799F36C480	Assembly-CSharp
AssignRandomTexture		240 B	6		Assembly-CSharp
AudioClipConfig		68.9 KB	304		Assembly-CSharp
AudioClipConfig[]		96 B	2		Assembly-CSharp
AudioCueConfig		7.7 KB	76		Assembly-CSharp
AudioCueConfig.Entry		12.1 KB	311		Assembly-CSharp
AudioCueConfig.Entry[]		6.4 KB	101		Assembly-CSharp
AudioInstanceLimitConfig		1.3 KB	18		Assembly-CSharp
AudioPrefs		64 B	2		Assembly-CSharp
AutoDestroy		0.7 KB	21		Assembly-CSharp
AverageVelocity	Hero (Start position (down)) [SOID:5281813]	72 B		0x2799F362F50	Assembly-CSharp
Award[]		0.6 KB	2		Assembly-CSharp
AwardCompleteScene		1.2 KB	13		Assembly-CSharp
AwardList		24 B		0x2799F43C3A0	Assembly-CSharp
AwardList.<>c		16 B		0x2798C8837E0	Assembly-CSharp
AwardList.JsonEntry[]		32 B		0x2798CFE9630	Assembly-CSharp
AwardManager.AwardManagerInternal	Zzz_AwardManagerInternal	32 B		0x2799F3F5AB0	Assembly-CSharp
AwardMenu	Awards	136 B		0x27996E3A480	Assembly-CSharp
AwardUnlockedMessage[]		192 B	2		Assembly-CSharp
BaseState[]		9.7 KB	82		Assembly-CSharp
Billboard		14.3 KB	306		Assembly-CSharp
Billboard.BillboardManagerInternal	Zzz_BillboardManagerInternal	40 B		0x27996F05840	Assembly-CSharp
BootingMenu.Entry[]		32 B		0x2798C87FD00	Assembly-CSharp
CameraDepthTextureMode	Camera	40 B		0x2798CF5CF00	Assembly-CSharp
CameraShake		1.2 KB	8		Assembly-CSharp

Name	Value	Type
_Awards	0x2799F3F5A50	System.Collections.Generic.List<Award>
_items	Award[64]	Award[]
_size	36	System.Int32
_version	36	System.Int32
_syncRoot	null	System.Object

Type	C++ Name	Depth
AwardManager	static_Instance	3
MonoBehaviour	Zzz_AwardManagerInternal	4

Type	C++ Name
System.Collections.Generic.List<Award>	

Type	C++ Name
..AwardManagerInternal	Zzz_AwardManagerInternal



































Left-click on a column to sort. Right-click on a column header to toggle individual columns.

C# Type	C++ Name	Size	Count	Address	Assembly
AwardCompleteScene		1.2 KB	13		Assembly-CSharp
AwardList		24 B		0x2799F43C3A0	Assembly-CSharp
AwardList.<>c		16 B		0x2798C8837E0	Assembly-CSharp
AwardList.JsonEntry[]		32 B		0x2798CFE9630	Assembly-CSharp
AwardManager.AwardManagerInternal	Zzz_AwardManagerInternal	32 B		0x2799F3F5AB0	Assembly-CSharp
AwardMenu	Awards	136 B		0x27996E3A480	Assembly-CSharp
AwardUnlockedMessage[]		192 B	2		Assembly-CSharp
BaseState[]		9.7 KB	82		Assembly-CSharp

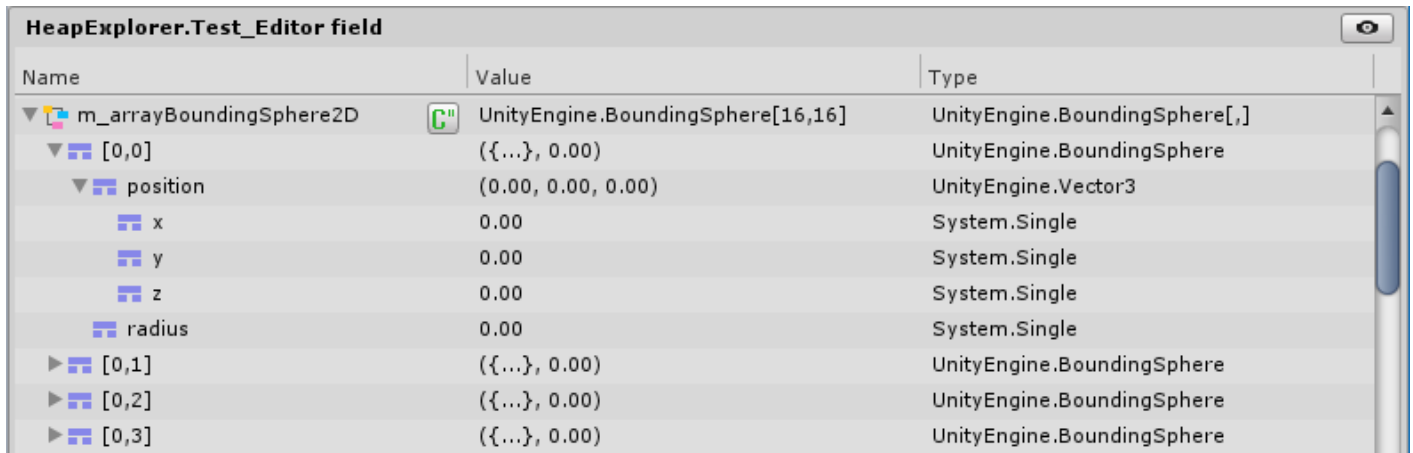
C# Type	The managed type of the object instance, such as System.String.
C++ Name	If the C# object has a C++ counter-part, basically C# types that derive from UnityEngine.Object have, the name of the C++ native object is displayed in this column (UnityEngine.Object.name).
Size	The amount of memory a managed object or group of managed objects is using.
Count	The number of managed objects in a group.
Address	The memory address of a managed object.
Assembly	The assembly name in which the type lives.

C# Object Inspector

The C# Object Inspector displays fields of a managed object, along with the field type and value. I tried to mimic the feel of Visual Studio's Watch window.

HeapExplorer.Test_Editor field			
Name	Value	Type	
 m_snapshot	null	HeapExplorer.PackedMemorySnapshot	
 m_intOne	1	System.Int32	
 m_intTwo	2	System.Int32	
 m_intThree	3	System.Int32	
 m_intFour	4	System.Int32	
▶  m_matrix	(1.00, 0.00, 0.00, 0.00, 0.00, 1.00, 0	UnityEngine.Matrix4x4	
▶  m_quaternion	(0.00, 0.00, 0.00, 1.00)	UnityEngine.Quaternion	
▶  m_quaternion1	(0.00, 0.71, 0.00, 0.71)	UnityEngine.Quaternion	
 m_decimal	1234567.89	System.Decimal	
 m_decimalNegative	-1234567.89	System.Decimal	
▶  m_arrayBoundingSphere2D	 UnityEngine.BoundingSphere[16,16]	UnityEngine.BoundingSphere[,]	
▶  m_arrayInt1D	 System.Int32[10]	System.Int32[]	
▶  m_arrayInt2D	 System.Int32[5,2]	System.Int32[,]	
▶  m_arrayInt3D	 System.Int32[3,2,2]	System.Int32[,,]	
▶  m_arrayJaggedInt2D	 System.Int32[][5]	System.Int32[][]	
▶  m_arrayJaggedInt3D	 System.Int32[][][3]	System.Int32[][][]	
▶  m_array2DJaggedVector2	 UnityEngine.Vector2[][5]	UnityEngine.Vector2[][]	
 m_arrayIntNull	null	System.Int32[]	
▶  m_vector2	(1.00, 2.00)	UnityEngine.Vector2	
▶  m_vector3	(1.00, 2.00, 3.00)	UnityEngine.Vector3	
▶  m_vector2int	(1, 2)	UnityEngine.Vector2Int	
▶  m_vector3int	(1, 2, 3)	UnityEngine.Vector3Int	
▶  m_genList	 0x42E047E0	System.Collections.Generic.List`1	
▶  m_rect	(1.00, 2.00, 3.00, 4.00)	UnityEngine.Rect	
▶  m_stringLong	 "Lorem ipsum dolor sit amet, consete	System.String	
 m_byteEnum	(0x1)	ByteEnum	
 m_uint16Enum	(0x2)	UInt16Enum	
 m_intEnum	(3)	IntEnum	
▶  m_refs	 RefType[4]	RefType[]	
▶  m_myClass	 0x393A0230	MyClass	
▶  m_myStruct	(1, 0)	MyStruct	
▶  m_vec2	 UnityEngine.Vector2[4]	UnityEngine.Vector2[]	
 m_magic0	0xDEADBEEF	System.UInt32	
▶  m_testString	 "This is a string"	System.String	

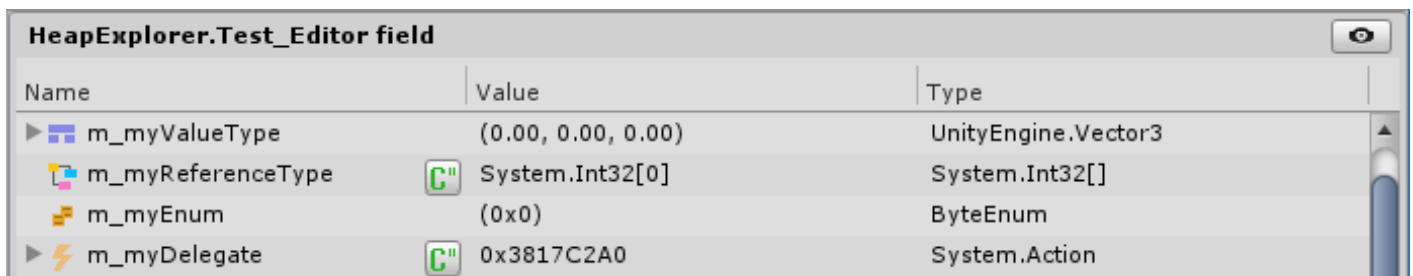
The arrow in-front of the Name indicates that the field provides further fields itself, or in the case of an array, provides array elements. Click the arrow to expand, as shown below.



The screenshot shows a window titled "HeapExplorer.Test_Editor field" with a table of fields. The field "m_arrayBoundingSphere2D" is expanded, showing its elements. The table has three columns: Name, Value, and Type.

Name	Value	Type
▼ m_arrayBoundingSphere2D	UnityEngine.BoundingSphere[16,16]	UnityEngine.BoundingSphere[,]
▼ [0,0]	({...}, 0.00)	UnityEngine.BoundingSphere
▼ position	(0.00, 0.00, 0.00)	UnityEngine.Vector3
x	0.00	System.Single
y	0.00	System.Single
z	0.00	System.Single
radius	0.00	System.Single
▶ [0,1]	({...}, 0.00)	UnityEngine.BoundingSphere
▶ [0,2]	({...}, 0.00)	UnityEngine.BoundingSphere
▶ [0,3]	({...}, 0.00)	UnityEngine.BoundingSphere

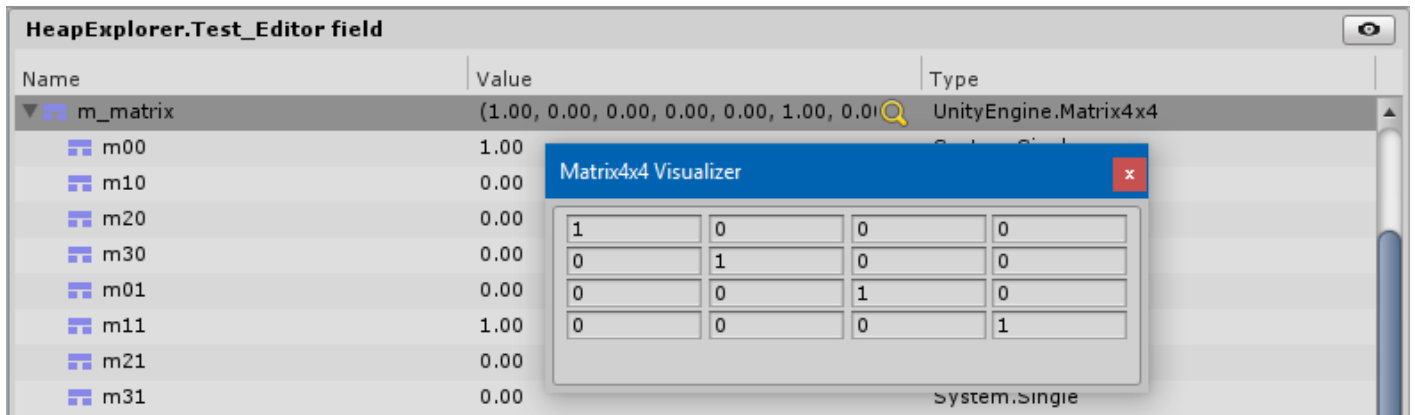
The icon in-front of the Name represents the “high-level type” of a field, such as: ReferenceType, ValueType, Enum and Delegate. If the field is a ReferenceType, a button is shown next to the Name, which can be used to jump to the object instance.



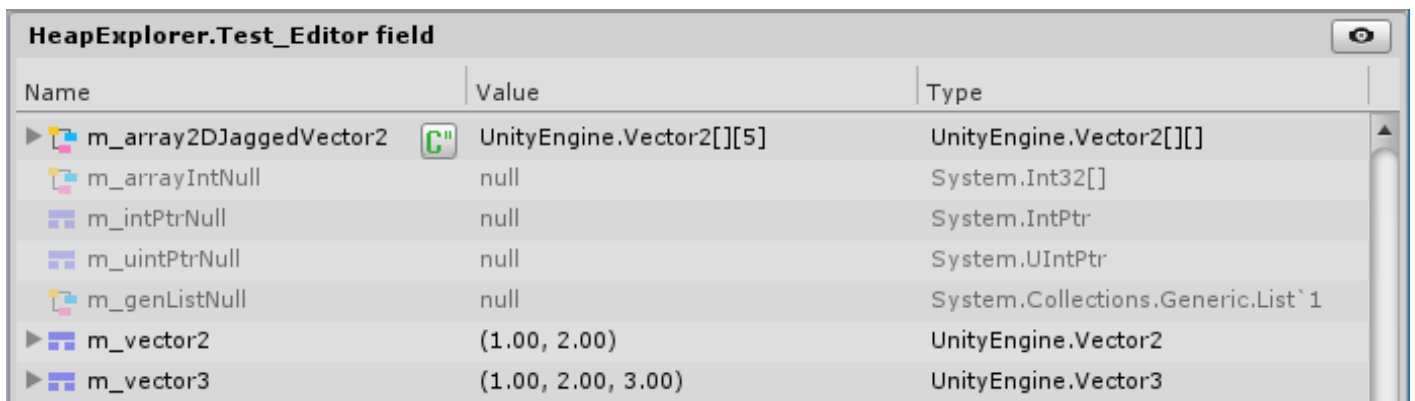
The screenshot shows a window titled "HeapExplorer.Test_Editor field" with a table of fields. The table has three columns: Name, Value, and Type.

Name	Value	Type
▶ m_myValueType	(0.00, 0.00, 0.00)	UnityEngine.Vector3
▶ m_myReferenceType	System.Int32[0]	System.Int32[]
▶ m_myEnum	(0x0)	ByteEnum
▶ m_myDelegate	0x3817C2A0	System.Action

A magnification icon appears next to the value, if the type provides a specific “Data Visualizer”. A data visualizer allows Heap Explorer to display the value in a more specific way, tailored to the type, as shown below.



If a field is a pointer-type (ReferenceType, IntPtr, UIntPtr), but it points to null, the field is grayed-out. I found this very useful, because you often ignore null-values and having those grayed-out, makes it easier to skip them mentally.











The eye-like icon in the top-right corner of the Inspector can be used to toggle between the field- and raw-memory mode. I don't know how useful the raw-memory mode is for you, but it helped me to understand object memory, field layouts, etc while I was developing Heap Explorer. I thought there is no need to remove it.

HeapExplorer.Test_Editor field																
Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000000003F2228C0	00	BB	22	37	00	00	00	00	00	00	00	00	00	00	00	00
000000003F2228D0	00	00	00	00	00	00	00	00	98	78	A4	4C	00	00	00	00
000000003F2228E0	A0	C2	17	38	00	00	00	00	00	50	18	38	00	00	00	00
000000003F2228F0	A0	55	BE	49	00	00	00	00	80	04	18	38	00	00	00	00
000000003F222900	80	5A	86	3A	00	00	00	00	20	F8	17	38	00	00	00	00
000000003F222910	80	3C	22	3F	00	00	00	00	E0	F6	17	38	00	00	00	00
000000003F222920	00	00	00	00	00	00	00	00	40	D7	33	4A	00	00	00	00
000000003F222930	00	70	1D	3F	00	00	00	00	40	F6	17	38	00	00	00	00
000000003F222940	20	78	A4	4C	00	00	00	00	10	54	BE	49	00	00	00	00
000000003F222950	40	43	BE	49	00	00	00	00	00	00	00	00	00	00	00	00
000000003F222960	00	00	00	00	00	00	00	00	01	00	00	00	02	00	00	00
000000003F222970	03	00	00	00	04	00	00	00	00	00	80	3F	00	00	00	00
000000003F222980	00	00	00	00	00	00	00	00	00	00	00	00	00	80	3F	
000000003F222990	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000003F2229A0	00	00	80	3F	00	00	00	00	00	00	00	00	00	00	00	00
000000003F2229B0	00	00	00	00	00	00	80	3F	00	00	00	00	00	00	00	00
000000003F2229C0	00	00	00	00	00	00	80	3F	00	00	00	00	F3	04	35	3F
000000003F2229D0	00	00	00	00	F3	04	35	3F	00	00	02	00	00	00	00	00
000000003F2229E0	15	CD	5B	07	00	00	00	00	00	00	02	80	00	00	00	00
000000003F2229F0	15	CD	5B	07	00	00	00	00	00	00	80	3F	00	00	00	40
000000003F222A00	00	00	80	3F	00	00	00	40	00	00	40	40	01	00	00	00
000000003F222A10	02	00	00	00	01	00	00	00	02	00	00	00	03	00	00	00
000000003F222A20	00	00	80	3F	00	00	00	40	00	00	40	40	00	00	80	40
000000003F222A30	01	00	02	00	03	00	00	00	01	00	00	00	00	00	00	00
000000003F222A40	EF	BE	AD	DE	00	00	00	00								

References / Referenced by

The “References” and “Referenced by” panel shows what objects are connected.

References to 3 object(s)			Referenced by 2 object(s)		
Type		C++ Name	Type		C++ Name
 MonoBehaviour	 	Zzz_PerkManagerInte	 PerkManager		static PerkManager._Instance
 PerkList			 GCHandle	 	PerkManagerInternal
 System.Collections.Generic.Dictionary`2					

For example, consider the following class:

```
class MeshFilter
{
    public Mesh mesh;
}
```

From the perspective of the MeshFilter, the “mesh” object is considered a “Reference”. From the perspective of the “Mesh”, the “MeshFilter” is shown as “Referenced by”.

Root Path

The Root Path panel is used to show the root paths of an object instance. A root path is the path of referrers from a specific instance to a root. A root can, for example, be a [Static Field](#), a [ScriptableObject](#), an [AssetBundle](#) or a [GameObject](#).

The root path can be useful for identifying memory leaks. The root path can be used to derive why an instance has not been garbage collected, it shows what other objects keep the instance in memory.

The Root Path View lists paths to static fields first, as those are often the cause why an instance has not been garbage collected. It then lists all paths to non-static fields. The list is sorted by depth, meaning shorter paths appear in the list first. Therefore, the "Shortest Path to Root" is shown at the top of the list.

2 Path(s) to Root			
Type	C++ Name		Depth
▼ S ▲ PerkManager	static _Instance		3
G" PerkManager.PerkManagerInternal	C+ C" Zzz_PerkManagerInternal		
G" System.Collections.Generic.Dictionary<System.Int32,System.Boolean>			
▼ C+ MonoBehaviour	G" C" Zzz_PerkManagerInternal		4
G" GCHandle	G" C+ PerkManager.PerkManagerInternal		
G" PerkManager.PerkManagerInternal	C+ C" Zzz_PerkManagerInternal		
G" System.Collections.Generic.Dictionary<System.Int32,System.Boolean>			
! Static fields are global variables. Anything they reference will not be unloaded.			

In the example above, Dictionary<Int32,Boolean> is kept in memory, because PerkManagerInternal holds a reference to it. Finally, the static field PerkManager holds a reference to the PerkManagerInternal object.

If you select a root path, the reason whether an object is kept in memory, is shown in the info message field at the bottom of the Root Path View.

Some types display a warning icon in the Root Path View. This is an indicator that this object is not automatically unloaded by Unity during a scene change for example.

Unity allows to flag UnityEngine.Object objects to prevent the engine from unloading objects automatically. This can be done, for example, using [HideFlags](#) or [DontDestroyOnLoad](#). The Root Path view displays a warning icon next to the type name, if an object is either a static field or uses one of Unity's mechanism to prevent it from being unloaded automatically.

C# Object Duplicates

The C# Object Duplicates View analyzes managed objects for equality. If at least two objects have identical content, those objects are considered duplicates.

13333 managed object duplicate(s) wasting 1.1 MB memory

C# Type	C++ Name	Size	Count	Address	Assembly
System.String		268 B		0x264AAAC0EAD	mscorlib
System.String		268 B		0x264AA4A5000	mscorlib
System.String		268 B		0x264AA4A5E00	mscorlib
System.String		268 B		0x264AA4A1000	mscorlib
System.String		268 B		0x264AA4A3D68	mscorlib
System.String		268 B		0x264AA4A1270	mscorlib
System.String		268 B		0x264AA4A14E0	mscorlib
System.String		268 B		0x264AA4A7AE0	mscorlib
System.String		268 B		0x264AA4A33A8	mscorlib
System.String		268 B		0x264AA4A3138	mscorlib
System.String		268 B		0x264AA4A3618	mscorlib
System.String		268 B		0x264AA4A3AF8	mscorlib
System.String		268 B		0x264AA4A3888	mscorlib
System.String		132.3 KB	513		mscorlib
UnityEngine.UIVertex[]		121.4 KB	8		UnityEngine.TextRenderingModule
System.Collections.Generic.List`1		91.1 KB	2914		mscorlib
UnityEngine.Vector3[]		55.2 KB	706		UnityEngine.CoreModule
System.Collections.Generic.List`1		43.1 KB	1378		mscorlib
UnityEngine.UIVertex[]		39.4 KB	120		UnityEngine.TextRenderingModule
UnityEngine.UIVertex[]		35.8 KB	4		UnityEngine.TextRenderingModule
System.Collections.Generic.List`1		29.6 KB	946		mscorlib
UnityEngine.UIVertex[]		25.1 KB	4		UnityEngine.TextRenderingModule
System.String		24.1 KB	101		mscorlib
System.String		19.5 KB	84		mscorlib
UnityEngine.UILineInfo[]		17.2 KB	50		UnityEngine.TextRenderingModule
StateTransition[]		15.9 KB	510		Assembly-CSharp
UnityEngine.UIVertex[]		15.5 KB	2		UnityEngine.TextRenderingModule
UnityEngine.Vector2[]		12.8 KB	3		UnityEngine.CoreModule
System.Collections.Generic.List`1		9.1 KB	291		mscorlib
System.Collections.Generic.List`1		7.5 KB	240		mscorlib

System.String field(s)

Name	Value	Type
length	123	System.Int32
start_char	'U'	System.Char

References to 0 object(s)

Type	C++ Name	Address
------	----------	---------

Referenced by 1 object(s)

Type	C++ Name	Address
CullStateChangedEvent		0x264A5064380

1 Path(s) to Root

Type	C++ Name	Depth
Text	Text	5
UnityEngine.UI.Text	UnityEngine.UI.Text	
CullStateChangedEvent		
System.String		

File root object selected.

The view groups duplicates by type. If a type, or group, occurs more than once in this view, it means it's the same type, but different content.

For example, if you have 10 “Banana” strings and 10 “Apple” strings, these would be shown as two “System.String” groups. Two string groups, because both are of the same type, but with different content.

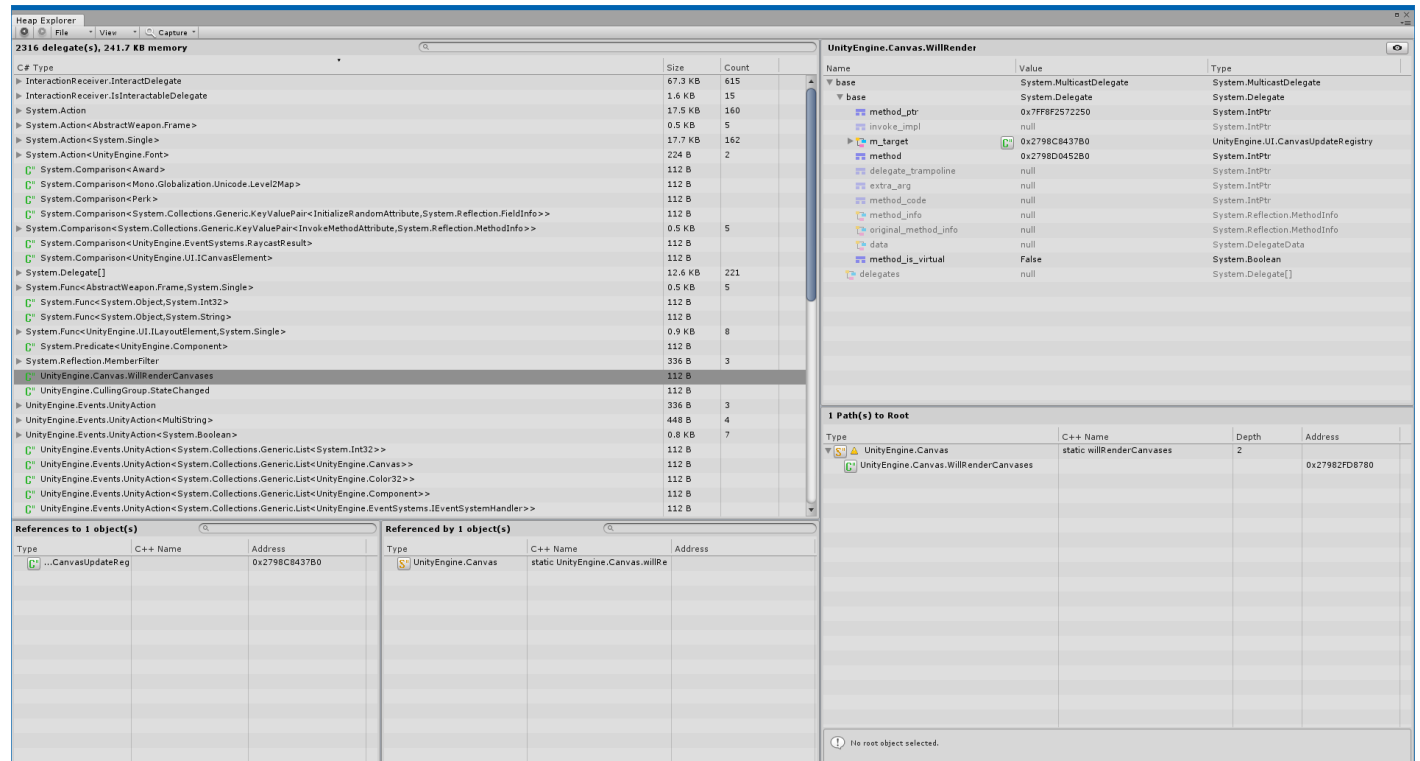
The view can be sorted by various columns. The most interesting ones likely being “Size” and “Count”.

Sorting the view by “Size” allows to quickly see where most memory is wasted due duplicated objects.

C# Delegates

Delegate's often seem to be the cause of a memory leak. I found it useful to have a dedicated view that shows all object instances that are of type System.Delegate.

The C# Delegates View is doing exactly this and behaves just like the regular [C# Objects view](#). It lists all object instances that are a sub-class of the System.Delegate type.



If you select a delegate, its fields are displayed in the inspector (top-right corner of the window) as shown in the image below.

UnityEngine.Canvas.WillRender		
Name	Value	Type
▼ base	System.MulticastDelegat	System.MulticastDelegate
▼ base	System.Delegate	System.Delegate
method_ptr	0x7FF8F2572250	System.IntPtr
invoke_impl	null	System.IntPtr
▶ m_target	0x2798C8437B0	UnityEngine.UI.CanvasUpdat
method	0x2798D0452B0	System.IntPtr
delegate_trampoline	null	System.IntPtr
extra_arg	null	System.IntPtr
method_code	null	System.IntPtr
method_info	null	System.Reflection.MethodInf
original_method_info	null	System.Reflection.MethodInf
data	null	System.DelegateData
method_is_virtual	False	System.Boolean
delegates	null	System.Delegate[]

“m_target” is a reference to the object instance that contains the method that is being called by the delegate.
“m_target” can be null, if the delegate points a static method.

Want to help?

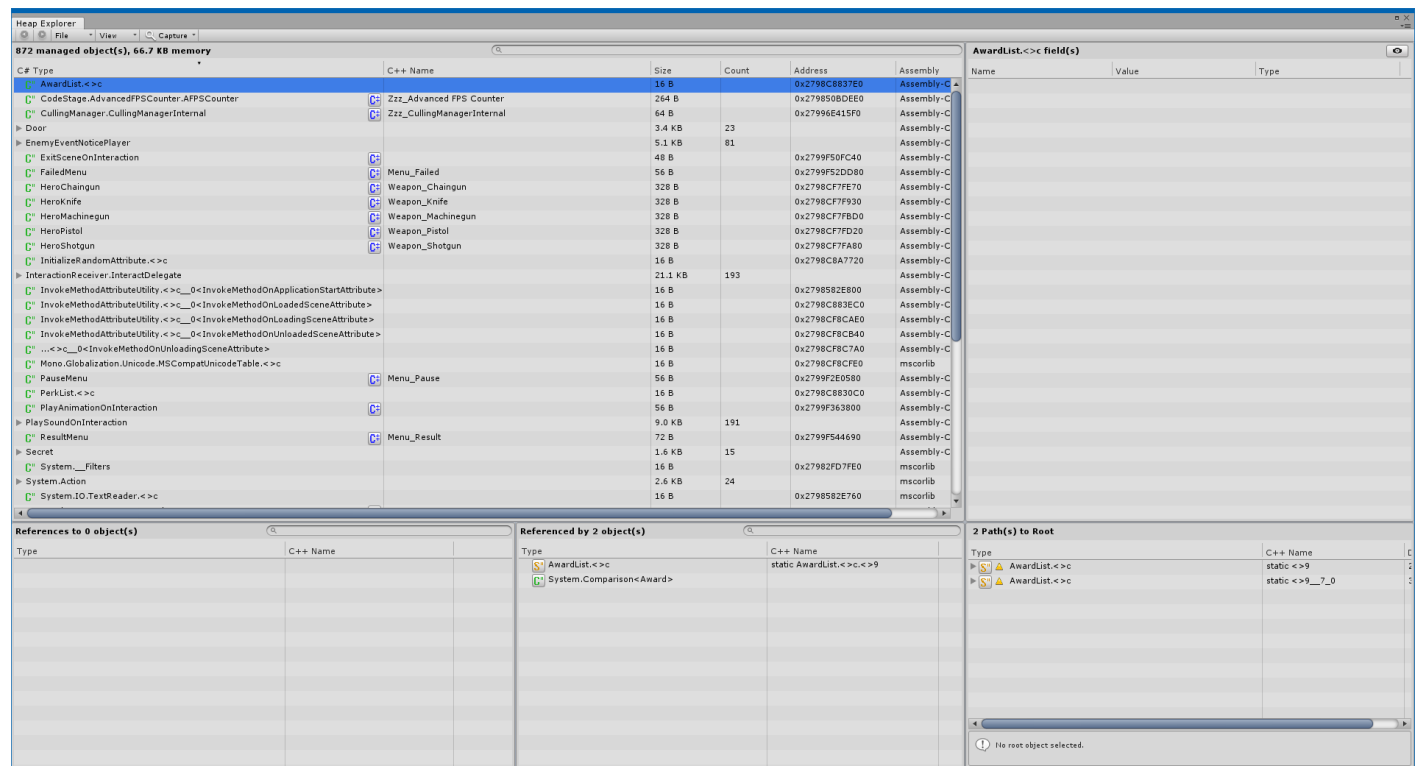
I would really like to display the actual method name of the field “method”. However, I didn’t find a way how I would look up the name using just its address. It would be a very useful feature imo. If you know how to do that, please let me know!

<https://forum.unity.com/threads/packedmemorysnapshot-how-to-resolve-system-delegate-method-name.516967/>

C# Delegate Targets

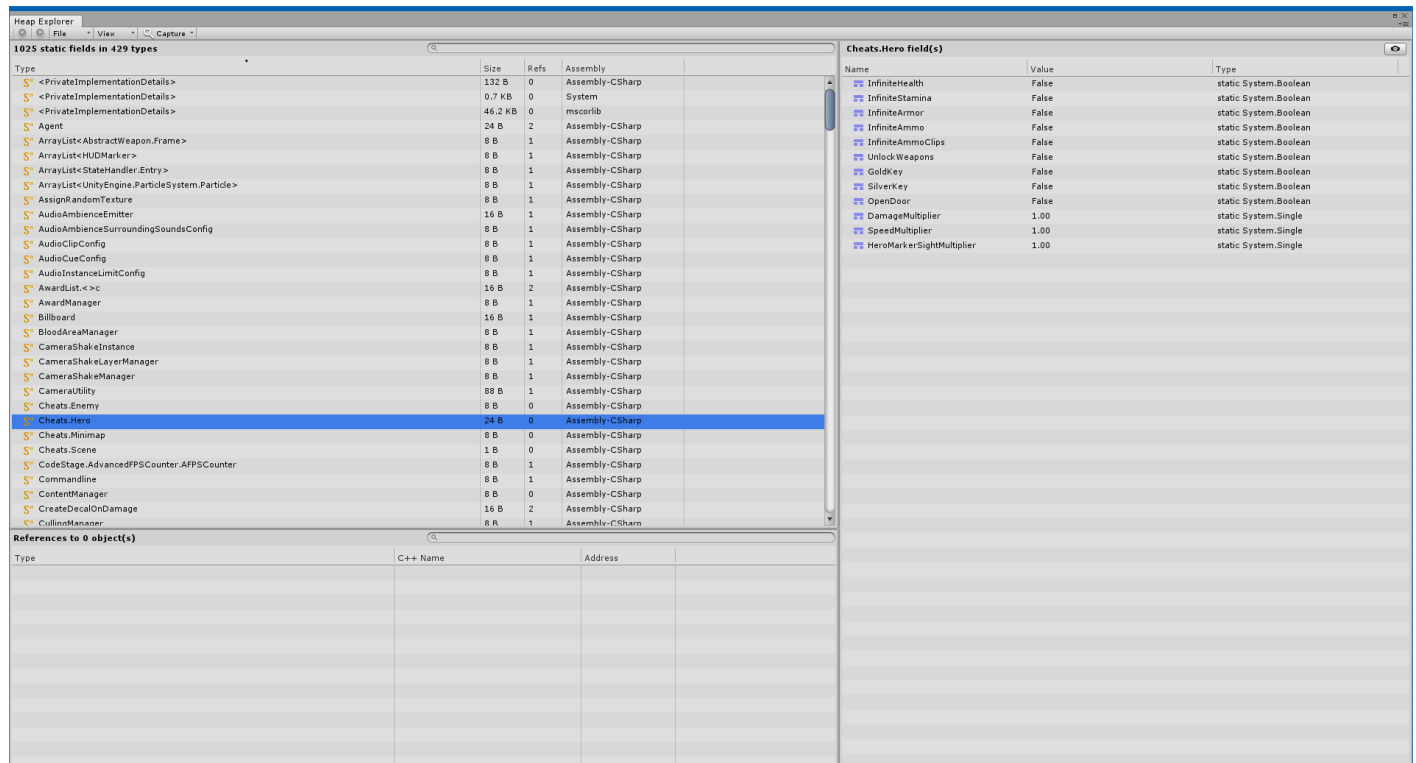
The C# Delegate Targets View displays managed objects that are referenced by the “m_target” field of a System.Delegate. The view behaves just like the regular [C# Objects view](#).

Having a dedicated Delegate Targets view allows to quickly see what objects are held by delegates.



C# Static Fields

The C# Static Fields view displays managed types that contain at least one static field. Selecting a type displays all of its static fields in the [inspector](#) (top-right corner).



Why is a static type missing?

According to my tests, static field memory is initialized when you first access a static type. If you have a static class in your code, but it is missing in the memory snapshot, it's likely that your application did not access this class yet.

Why does it not display an “Address” column?

Unity's MemorySnapshot API does not provide at which memory address static field data is located.

Where is the Root Path view?

Static fields itself represent a root. There is no need for the Root Path view, because every static type is a root object.

Where is the “Referenced By” view?

You can't reference a static field. However, static fields can reference other objects, that's why it shows the "References" view.

C# Memory Sections

The C# Memory Sections view displays heap sections found in the memory snapshot.

This view shows how many of those memory sections exist, which gives you an idea of how memory is fragmented. Select a memory section in the left list, to see what objects the sections contains, which are shown in the list on the right.

Heap Explorer

File

View

Capture

11 Managed Heap Section(s), using 9.6 MB, fragmented across 463.3 MB

Address	Size
0x2799F80000	1.9 MB
0x2799F21000	1.4 MB
0x27996E0000	1.2 MB
0x2799F80000	1.8 MB
0x2798CF0000	0.9 MB
0x2799FEA800	372.0 KB
0x2798C81000	0.6 MB
0x27985F0000	420.0 KB
0x2798582000	0.5 MB
0x2798509000	256.0 KB
0x27982F8000	256.0 KB

References to 364 object(s)

Type	C++ Name	Address
System.String		0x27982FB4F40
UnityEngine.UI.CanvasScaler	Weapon_Machinegun	0x27982FB5000
UnityEngine.UI.CanvasScaler	Vignette	0x27982FB5060
UnityEngine.UI.CanvasScaler	Crosshair_Gun_Heavy	0x27982FB50C0
UnityEngine.UI.CanvasScaler	Weapon_Knife	0x27982FB5120
UnityEngine.UI.CanvasScaler	Health	0x27982FB5180
UnityEngine.UI.CanvasScaler	Ammo	0x27982FB51E0
UnityEngine.UI.CanvasScaler	Interaction	0x27982FB5240
UnityEngine.UI.CanvasScaler	Weapon_Pistol	0x27982FB52A0
UnityEngine.UI.CanvasScaler	Weapon_Chainingun	0x27982FB5300
UnityEngine.UI.CanvasScaler	Lighting	0x27982FB5360
UnityEngine.UI.CanvasScaler	Saving	0x27982FB53C0
UnityEngine.UI.CanvasScaler	Armor	0x27982FB5420
UnityEngine.UI.CanvasScaler	Perk	0x27982FB5480
HUdPerk	Perk	0x27982FB54E0
UnityEngine.UI.CanvasScaler	Keys	0x27982FB5540
UnityEngine.UI.CanvasScaler	Crosshair_Knife	0x27982FB55A0
UnityEngine.UI.CanvasScaler	Minimap	0x27982FB5600
...Generic.Dictionary<UnityEngine.Transform,UnityEngine.UI.Image>		0x27982FB5660
DynamicMaterial	Sprite	0x27982FB56C0
DynamicMaterial	Sprite	0x27982FB5720
DynamicMaterial	Sprite	0x27982FB5780
DynamicMaterial	Sprite	0x27982FB57E0
DynamicMaterial	Sprite	0x27982FB5840
DynamicMaterial	Sprite	0x27982FB58A0
DynamicMaterial	Sprite	0x27982FB5900
DynamicMaterial	Sprite	0x27982FB5960
DynamicMaterial	Sprite	0x27982FB59C0
DynamicMaterial	Sprite	0x27982FB5A20
DynamicMaterial	Sprite	0x27982FB5A80
DynamicMaterial	Sprite	0x27982FB5AE0
DynamicMaterial	Sprite	0x27982FB5B40
DynamicMaterial	Sprite	0x27982FB5BA0
DynamicMaterial	Sprite	0x27982FB5C00
DynamicMaterial	Sprite	0x27982FB5C60
DynamicMaterial	Sprite	0x27982FB5CC0
DynamicMaterial	Sprite	0x27982FB5D20
DynamicMaterial	Sprite	0x27982FB5D80
DynamicMaterial	Sprite	0x27982FB5DE0
...Generic.Dictionary<System.String,System.LocalDataStoreSlot>		0x27982FB5E40
System.String		0x27982FB640
AudioCueConfig.Entry[]		0x27982FB73C0
AudioCueConfig.Entry[]		0x27982FB7640
AudioCueConfig.Entry[]		0x27982FB7940
System.Delegate[]		0x27982FB7B80
System.Object[]		0x27982FB7C80
System.Object[]		0x27982FB7CC0

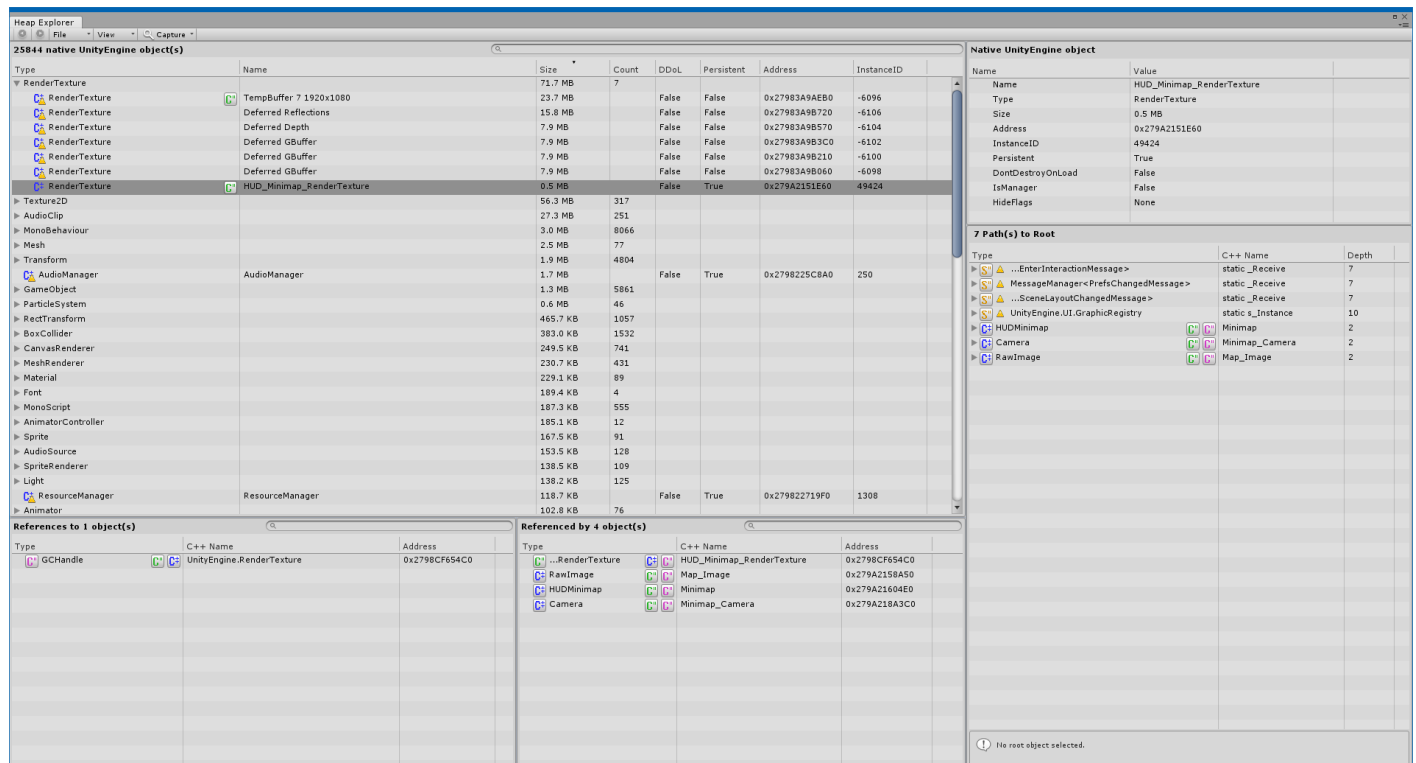
C++ Objects

The C++ Objects view displays native UnityEngine objects found in a memory snapshot.

Unity offers little information of native objects, unfortunately. While it does provide the size of objects, a lot of data that would be interesting is missing, such as the width/height of a texture for example.

The view has pretty much the same features as the [C# Objects view](#), but it does not provide functionality to inspect native object properties, beside the few ones Unity provides.

The view features the main list that contains all native UnityEngine objects (top-left), the limited information about the selected object in the top-right panel, [root paths](#) (bottom-right) and which objects it [references and is referenced by](#).



Here is what the C++ objects view displays.

Name	Description
Type	The type of a native object.
Name	The name of anative object. This what you can query using UnityEngine.Object.name

Size	The size of a native UnityEngine.Object that it consumes in memory.
Count	The number of native objects of the same type.
DDoL	Has this object has been marked as DontDestroyOnLoad?
Persistent	Is this object persistent? (Assets are persistent, objects stored in scenes are persistent, dynamically created objects are not)
Address	The memory address of the native C++ object. This matches the "m_CachedPtr" field of UnityEngine.Object.
InstanceID	The UnityEngine.Object.GetInstanceID() value. From my observations, positive id's indicate persistent objects, while negative id's indicate objects created during runtime.
IsManager	Is this native object an internal Unity manager object?
HideFlags	The UnityEngine.Object.hideFlags this native object has.

Major Unity bug?!

Native objects often contain hundreds of millions of references to thousands of completely unrelated objects. I'm pretty sure this a bug in the Unity editor or engine and reported it as Case 987839:

<https://issuetracker.unity3d.com/issues/packedmemorysnapshot-unexpected-connections-between-native-objects>

The following forum post, shows an actual real example of the problem. Unity creates a connection array of 509.117.918 elements.

<https://forum.unity.com/threads/wip-heap-explorer-memory-profiler-debugger-and-analyzer-for-unity.527949/page-2#post-3617188>

Here is another post in the Profiler forum, which sounds pretty much like the same issue.

<https://forum.unity.com/threads/case-1115073-references-to-native-objects-that-do-not-make-sense.608905/>



Exclude NativeObject connections

In order to workaround the major connections bug I described in the read box above this text, where Unity provides more than five hundred million connections, I implemented a feature in Heap Explorer to exclude native object connections.

Unity still captures those connections, but Heap Explorer will not process most of them. While this means Heap Explorer isn't able to show which native object are connected, it's perhaps still better than not being able to capture a snapshot at all.

You can active this option from Heap Explorer's "File > Settings" menu. The option is called "Exclude NativeObject connections".

Only active this option, if you see Heap Explorer running in this error:

HeapExplorer: Failed to allocate 'new PackedConnection[]'.

C++ Asset Duplicates (guessed)

The “C++ Asset Duplicates” view is guessing which asset might be duplicates. Unity Technologies explains how to guess if two assets are duplicates on this page:

<https://docs.unity3d.com/Manual/BestPracticeUnderstandingPerformanceInUnity2.html>

The screenshot shows the Heap Explorer interface with the following sections:

- C++ Asset Duplicates (guessed)**: A table listing assets with columns: Type, Name, Size, Count, DDoL, Persistent, Address, and InstanceID.
- Native UnityEngine object**: A table showing details for a selected object, including Name, Type, Size, Address, InstanceID, Persistent, DontDestroyOnLoad, IsManager, and HideFlags.
- 15 Path(s) to Root**: A table showing the path of the object to the root, including Type, C++ Name, Depth, and Address.
- References to 1 object(s)**: A table showing references to the selected object, including Type, C++ Name, and Address.
- Referenced by 3 object(s)**: A table showing objects that reference the selected object, including Type, C++ Name, and Address.

At the bottom, a status bar indicates: 2 native UnityEngine object guessed duplicate(s) wasting 128.6 KB memory.

The idea is, if you have two assets of the same type and name, but with different instanceID's, these could be duplicates. This approach is of course everything else than reliable, but perhaps better than nothing. Therefore, I implemented this view in Heap Explorer.

The assumption falls apart as soon as the project contains more than one asset with the same name of the same type. For example, if you follow a project structure like shown below, Heap Explorer shows incorrect results.

Asset Duplicate Detection Limitation

Heap Explorer incorrectly detects these textures as duplicates, because they have the same name:

Assets/Characters/Alien_01/Texture.tga

Assets/Characters/Alien_02/Texture.tga

Assets/Characters/Alien_02/Texture.tga

Message to Unity Technologies

It would be very useful if you store additional information in a “development mode” build, to be able to map a native UnityEngine object in a MemorySnapshot to its asset guid in the project for example. This would allow tools such as Heap Explorer to implement some powerful functionality. This additional information should be stripped in non-development builds.

Observations

In this section, I'm going to list a few things I observed while working on Heap Explorer and studying many memory snapshots.

String duplicates

Unity does not seem to share string memory.

Even if two strings have the same text, the text exists in memory twice. This is very noticeable in various UnityEngine.UI Components, that seem to use strings a lot, such as function binding for example.

I found many UI heavier games contain thousands of string duplicates that all originate from Unity's UI system, such as:

```
UnityEngine.UI.MaskableGraphic+CullStateChangedEvent, UnityEngine.UI, Version=1.0.0.0,  
Culture=neutral, PublicKeyToken=null
```

Zero Size Array (new T[0])

I found many projects contain fields of type List<T> and T[], but those collections often do not contain any elements, they have a size of 0.

Fields exposed to the Inspector seem to be created by Unity automatically. I guess, it's a feature to allow you to edit the value, even if you do not create that object instance yourself in code.

However, what this is causing is that you might believe these fields are null and don't consume any memory.

In reality, projects often seem to contain thousands of zero size array's (new T[0]). Please be aware that an empty array still consumes memory, due to its header. If you have thousands of zero size array's, you might waste memory.

Some famous asset store plugin(s) pull in tens-of-thousands of empty arrays.

UnityEngine.Events.UnityEventBase

In UI heavier projects, this type seems to cause a lot of string duplicates and zero size arrays.

It contains an `InvokableCallList` field named `m_Calls`, which itself contains three `List<>`'s (`m_PersistentCalls`, `m_RuntimeCalls`, `m_ExecutingCalls`), which seem in most cases to have no elements.

The `m_TypeName` string field points in many cases to duplicated strings.

UnityEngine.Events.UnityEvent			
Name	Value	Type	
▼ base	UnityEngine.Events.UnityEventBase	UnityEngine.Events.UnityEventBase	
▼ m_Calls	0x2799F301280	UnityEngine.Events.InvokableCallList	
▼ m_PersistentCalls	0x2799F268B10	System.Collections.Generic.List<UnityEngine.Eve	
_items	UnityEngine.Events.BaseInvokableCall[0]	UnityEngine.Events.BaseInvokableCall[]	
_size	0	System.Int32	
_version	1	System.Int32	
_syncRoot	null	System.Object	
▼ m_RuntimeCalls	0x2799F268AE0	System.Collections.Generic.List<UnityEngine.Eve	
_items	UnityEngine.Events.BaseInvokableCall[0]	UnityEngine.Events.BaseInvokableCall[]	
_size	0	System.Int32	
_version	0	System.Int32	
_syncRoot	null	System.Object	
▼ m_ExecutingCalls	0x2799F268AB0	System.Collections.Generic.List<UnityEngine.Eve	
_items	UnityEngine.Events.BaseInvokableCall[0]	UnityEngine.Events.BaseInvokableCall[]	
_size	0	System.Int32	
_version	0	System.Int32	
_syncRoot	null	System.Object	
m_NeedsUpdate	True	System.Boolean	
▶ m_PersistentCalls	0x27996F208A0	UnityEngine.Events.PersistentCallGroup	
▶ m_TypeName	"UnityEngine.Events.UnityEvent, Unity	System.String	
m_CallsDirty	True	System.Boolean	
m_InvokeArray	null	System.Object[]	

Known Unity Issues

While working on Heap Explorer, I found various issues in Unity's Memory Profiling API:
<https://docs.unity3d.com/ScriptReference/MemoryProfiler.MemorySnapshot.html>

These issues are not limited to Heap Explorer, they are Unity editor/engine problems. Every tool, such as Unity's own Memory Profiler, is suffering from these issues as well.

The "Status" column indicates with which version I retested the issue after I submitted the original bug-report, which was confirmed as a bug by Unity Technologies.

Status	Case	Description	Link
2018.3.0f2 = FAIL	1115544	Array type 'baseOrElementTypeIndex' does not point to its element type, when using .NET 4.x Scripting Runtime.	https://forum.unity.com/threads/case-1115544-array-type-baseorelementtypeindex-does-not-point-to-its-element-type.609922/
2018.3.0b11 = FAIL Closed bug with "By Design"	1104590	"subSystemList" detected with wrong type	https://forum.unity.com/threads/case-1104590-packedmemorysnapshot-subsystemlist-detected-with-wrong-type.589420/
2017.4.10f1 = FAIL Closed bug with "Duplicate" without providing a link to the duplicate.	1104581	Missing "subSystemList" field	https://forum.unity.com/threads/case-1104581-packedmemorysnapshot-missing-subsystemlist-field-in-playerloopssystem-typedescripti.589405/
2017.4.10f1 = FAIL 2018.3.f02 = OK	1079363	typeDescriptions, gcHandles, managedHeapSections arrays are empty in memory snapshot when using .NET 4.x Scripting Runtime.	https://issuetracker.unity3d.com/product/unity/issues/guid/1079363
2017.4.6f1 = OK	1034172	"typeDescriptions" array is empty in memory snapshot when using .NET 3.5 Scripting Runtime.	https://issuetracker.unity3d.com/issues/snapshot-dot-typedescriptions-dot-length-returns-zero-when-taking-a-packedmemorysnapshot-snapshot

2017.4.6f1 = OK	973872	Fields of generic types, such as List<T>, are missing in memory snapshot.	
2017.4.6f1 = OK	974031	Leading period symbol in Typename	https:// issuetracker.unity3d.com/ issues/ packedmemorysnapshot- leading-period-symbol-in- typename
2017.4.6f1 = OK	974042	Nested types lack type of parent	https:// issuetracker.unity3d.com/ issues/ packedmemorysnapshot- nested-types-lack-type-of- parent
2017.4.6f1 = FAIL	975832	Missing connections between native objects	https:// issuetracker.unity3d.com/ issues/ packedmemorysnapshot- missing-connections- between-native-objects
2017.4.6f1 = OK	977003	Unable to resolve typeDescription of GCHandle.target	https:// issuetracker.unity3d.com/ issues/ packedmemorysnapshot- unable-to-resolve- typedescription-of- gchandle-dot-target
FAIL	977938	RequestNewSnapshot freezes editor for some time	https:// issuetracker.unity3d.com/ issues/ packedmemorysnapshot- requestnewsnapshot- freezes-editor-for-a-bit
2018.2.0a3 = OK 2017.4.6f1 = FAIL	983804	ScriptableObject recognized as MonoBehaviour	https:// issuetracker.unity3d.com/ issues/scriptableobject-is- recognized-as- monobehaviour-in-the- snapshot-dot-nativetypes- array
2017.4.6f1 = FAIL	984330	Type contains staticFieldBytes memory, but	https://

		actually has no static field	issuetracker.unity3d.com/issues/packedmemorysnapshot-type-contains-staticfieldbytes-but-has-no-static-field
2017.4.6f1 = OK	984704	Incorrect name for generic types	https://issuetracker.unity3d.com/issues/packedmemorysnapshot-incorrect-name-for-generic-types
FAIL BY DESIGN? I believe Unity didn't invest enough time to understand the problem.	987839	Unexpected connections between native objects. The "connections" array in a snapshot holds connections between native objects that hardly make sense, such as an "AudioSource" connects to a "Mesh". https://forum.unity.com/threads/wip-heap-explorer-memory-profiler-debugger-and-analyzer-for-unity.527949/page-2#post-3619123	https://issuetracker.unity3d.com/issues/packedmemorysnapshot-unexpected-connections-between-native-objects
2017.4.6f1 = FAIL	987987	"snapshot.managedHeapSections" do not match stats in Profiler	https://issuetracker.unity3d.com/issues/packedmemorysnapshot-managedheapsections-do-not-match-stats-in-profiler
2018.2.0b9 = FAIL 2017.4 = No NativeArray support	989622	NativeArray memory missing in snapshot	https://issuetracker.unity3d.com/issues/packedmemorysnapshot-nativearray-memory-is-not-captured-in-it
2017.4.6f1 = FAIL	989625	ExecutableAndDlls memory is missing in snapshot	https://issuetracker.unity3d.com/issues/executableanddlls-is-not-found-when-checking-through-a-packedmemorysnapshot
2017.4.6f1 = OK	993250	Calling RequestNewSnapshot method causes Player to crash	

2017.4.6f1 = FAIL	993295	Various fields are missing in types when capturing a snapshot from a Player. It seems many fields of an enum type are missing.	https:// issuetracker.unity3d.com/ issues/ packedmemorysnapshot- missing-fields-in-player
2017.4.6f1 = FAIL	998707	"delegates" field missing in "MulticastDelegate" type	https:// issuetracker.unity3d.com/ issues/ packedmemorysnapshot- delegates-field-missing-in- multicastdelegate-type
2017.4.6f1 = FAIL	1008321	Editor freezes if Profiler is unable to connect to Player	https:// issuetracker.unity3d.com/ issues/editor-freezes-if- profiler-is-unable-to- connect-to-player
2017.4.6f1 = OK	1008392	Editor crash while taking PackedMemorySnapshot in "CopyHeapSection"	https:// issuetracker.unity3d.com/ issues/editor-crash-while- taking- packedmemorysnapshot- in-copyheapsection
2017.4.6f1 = OK	998728	Editor window becomes unresponsive when using UnityEditor.IMGUI.Controls.TreeView for a while	https:// issuetracker.unity3d.com/ issues/unityeditor-dot- imgui-dot-controls-dot- treeview-becomes- unresponsive-if-it- contains-many-items
2018.2.0b9 = FAIL 2017.4.6f1 = FAIL	1052281	Keyboard navigation in UnityEditor.IMGUI.Controls.TreeView becomes unresponsive if the treeview contains many items	https://forum.unity.com/ threads/case-1052281- imgui-controls-treeview- keyboard-navigation-is- slow.537468/
2018.2.0b9 = FAIL 2017.4.6f1 = FAIL	1052308	nativeObject.isPersistent flag or its documentation is wrong	https://forum.unity.com/ threads/case-1052308- packedmemorysnapshot- nativeobject-ispersistent- or-documentation- incorrect.537484/

Empty C# Memory Sections

Some memory sections in the [C# Memory Sections](#) view do not contain any references to any objects, but if you look at their raw-memory, you see that it actually contains non-zero content.

This is caused by [C# Static Fields](#) memory being stored in managed heap sections as well. Unfortunately, the information Unity provides regarding [staticFieldBytes](#) is very limited in their Memory Profiling API.

A PackedMemorySnapshot contains for every managed type an array, that represents the memory that is used to store the static fields memory, named [staticFieldBytes](#). Unfortunately, it does not provide at which address in the managed heap this would be located.

Additionally, the staticFieldBytes are also stored across the [managed memory sections](#).

Due to missing functionality in the PackedMemorySnapshot, I'm unable to resolve what staticFieldBytes are stored in which MemorySection.

Which means, if a MemorySection contains staticFieldBytes, I'm unable to detect this and can't visualize it in Heap Explorer.

I haven't submitted a bug-report for this yet.

Feedback & Questions on Unity Memory Profiling API

PackedMemorySnapshot: Missing Unity version and platform information

<https://feedback.unity3d.com/suggestions/packedmemorysnapshot-missing-unity-version-and-platform-information>

MemoryProfiler: Unexpected NativeUnityEngineObject size

<https://forum.unity.com/threads/memoryprofiler-unexpected-nativeunityengineobject-size.519715/>

PackedMemorySnapshot: How to resolve System.Delegate method name?

<https://forum.unity.com/threads/packedmemorysnapshot-how-to-resolve-system-delegate-method-name.516967/>

Changelog

Beta 3.1 (February 13th, 2019)

- The bottom status bar now display the loaded snapshot filepath. Suggested [here](#).

Beta 3.0 (January 7th, 2019)

- Added option to hide internal managed managed sections (sections that are not 4096bytes aligned), as explained [here](#). You can toggle this behavior in the toolbar “File > Settings > Show Internal Memory Sections”.
- Added functionality to substitute managed object addresses from a text file to Heap Explorer. This was an attempt to debug why a managed object exists in memory at run-time, but is not included in a memory snapshot. You can follow the problem [here](#).

Beta 2.9 (November 28th, 2018)

- Added workaround for Unity bug [Case 1104590](#). Heap Explorer now ignored what appears as nested struct instances, which was causing an endless loop. This issue occurs with .NET4 ScriptingRuntime only. You can find the option to toggle this behavior in HeapExplorer toolbar > File > Settings > Ignore Nested Structs. Based on [this feedback](#).
- Increased loop guard limit, when trying to find root paths from 100000 to 300000 iterations. This avoids [this issue](#).
- Added functionality to cancel a “finding root paths” operation. A “Cancel” button is now available in the root paths panel.
- When changing the selection in the C#/C++ objects view, any running “finding root paths” job is aborted, rather than waited for completion. This makes the UI feel more responsive.
- Changed order in which various jobs run in Heap Explorer, which makes the UI slightly more responsive.

Beta 2.8 (November 25th, 2018)

- Added functionality to stop/cancel a processing step when Heap Explorer is analyzing a memory snapshot. This is actually a workaround for an issue that sounds like an infinite loop as [reported here](#). I’m still interested in the memory snapshot that causes the issue to actually fix it though.
- C# Memory Sections View: Added raw memory view to inspect the actual bytes of a memory section.
- C# Memory Sections View: Changed phrase "N memory sections fragmented across X.XXGB" to "N memory sections within an X.XXGB address space", based on [this feedback](#).
- Overview View: Removed memory sections graph, based on [this feedback](#).
- Added functionality to export parts of the memory snapshot to CSV. You find it in the “Views” popup menu, named “CSV Export”. Based on [this feedback](#).

Beta 2.7 (September 9th, 2018)

- Added warning if project uses .NET 4.x Scripting Runtime, because Unity's MemoryProfiling API does not work with that at the time of writing. [See here for details](#).
- Added better error messages for known memory snapshot issues, like empty typeDescriptions array.
- Added error message dialog and then quit Heap Explorer if an unrecoverable error occurs.
- Added error message if Heap Explorer is unable to parse the editor version, to get at the bottom of [this problem](#).

Beta 2.6 (September 2nd, 2018)

- Added "Exclude NativeObject connections when capturing a memory snapshot" option to Heap Explorers "File > Settings" menu. Please read the "[Exclude NativeObject connections](#)" documentation when you might want to activate it.

Beta 2.5 (August 28th, 2018)

- Added Debug.Log if the PackedConnection array creation would cause an out-of-memory exception. [See this forum post](#).

Beta 2.4 (August 11th, 2018)

- Overview View: Fixed negative size display if object group exceeded the maximum int limit.
- C++ Objects View: Fixed negative size display, if an object group exceeded the maximum int limit.
- C# Objects View: Fixed negative size display, if an object group exceeded the maximum int limit.
- Memory Sections: Fixed saving memory sections as file which are larger than 2gb.
- Loading memory snapshots displays more information about the progress now.

Alpha 2.3 (June 7th, 2018)

- C++ Objects View: Added asset preview. Memory snapshots do not contain asset memory, thus the preview is generated from the asset in the project instead rather than the memory snapshot!
- C++ Objects View: Reworked list filtering. It allows to change filter settings without the list being recreated every time you change a single option. You have to apply those changes now. This helps to save time when filtering big memory snapshots.
- C++ Objects View: If any list filtering is active, the "Filter" button in the toolbar uses a different color, to make it obvious filtering is in place.
- C++ Objects View: Fixed "Type" display of MonoBehaviour, which now displays the actual derived type rather than just MonoBehaviour always.
- C++ Objects View: Fixed "Type" column sometimes showing the object name rather than type.
- Brief Overview: Changing wording of the Memory Section description.

Alpha 2.2 (June 23rd, 2018)

- C++ Objects View: Replaced generic “C++” icon with icons that indicate whether it’s an asset, scene-object or run-time-object.
- C++ Objects View: Added popup menu to toolbar to exclude native objects from the list, depending on whether the native object is an asset, a scene-object or a run-time-object.
- C++ Objects View: Added popup menu to toolbar to exclude native objects from the list, depending on whether the native object is marked as “Destroy on load” or “Do NOT destroy on load”.
- C++ Objects View: Display icon in object inspector in the top-right corner.
- C++ Objects View: Fixed count and size display in bottom status bar.

Alpha 2.1 (June 16th, 2018)

- Added “C++ Asset Duplicates” view.
- Added context menu if you right-click the “Name” column in the C++ objects view. It provides functionality to find assets of the C++ object/asset name in the project,

Alpha 2.0 (May 25th, 2018)

- C# Memory Sections: Added visualization of managed memory section fragmentation

Alpha 1.9 (May 21st, 2018)

- Overview: Brought back “GCHandles” and “Virtual Machine Information”
- When loading a snapshot from the “Start Page”, it opens the “Brief Overview” afterwards, rather than the “C# Objects” view.

Alpha 1.8 (May 21st, 2018)

- Overview: Added percentage % next to the size
- Overview: Added visualization of managed heap fragmentation in the operating system memory
- Overview: Fixed some layout issues
- Overview: Removed GCHandles and VirtualMachine Information from the overview, as this wasn’t very interesting nor something that is normally using much memory.

Alpha 1.7 (May 19th, 2018)

- Compare Snapshots: Added “Swap” button to swap snapshot A <> B as suggested [here](#).
- Compare Snapshots: Snapshot B doesn’t get unloaded anymore, when loading or capturing snapshot A, as suggested multiple times.
- Compare Snapshots: Loading Snapshot B now displays the same loading status information as when loading Snapshot A.
- C# Static Fields: Added toolbar menu with functionality to save the memory of a selected static field as a file.

- C# Memory Sections: Added toolbar menu with functionality to save the all C# memory sections as a file.
- Moved statics to bottom status bar.
- Debugged why some memory sections do not contain references to managed objects, but contain non-zero bytes. See “Empty C# Memory Sections” in under [Known Issues](#).

Alpha 1.6 (May 8th, 2018)

- “Compare Snapshots”, changed diff from “A - B” to “B - A” comparison. Reported [here](#).
- Fixed being unable to select a memory section by clicking on its address in the “C# Memory Sections” view. Reported [here](#).
- Loading a snapshot restores the previously active view when done. Reported [here](#).

Alpha 1.5 (May 7th, 2018)

- Added another workaround for “Array out of index” exception.

Alpha 1.4 (May 7th, 2018)

- Added workaround for “Array out of index” exception, if the element-type of an array could not be detected. In this case this object outputs an error during heap reconstruction and from then on it’s ignored. Please send me the memory snapshot if you run into this error.

Alpha 1.3 (May 6th, 2018)

- In Heap Explorer toolbar, added “Capture > Open Profiler”. This opens the Unity profiler window, which allows you to connect to a different target. It’s simply a convenience feature, if you figure the editor is not connected to the correct player. This saves a few mouse clicks to get to the Unity Profiler to connect to a different player.
- The “Load” button in the “Compare Snapshot” view now features a “most recently used” snapshot list. This allows you to switch between previously saved snapshots with fewer mouse clicks.
- In Heap Explorer toolbar, added “File > Recent”, which allows to re-open the “most recently used” snapshots.

Alpha 1.2 (May 5th, 2018)

- In Heap Explorer toolbar, renamed “Capture” to “Capture and Analyze”.
- In Heap Explorer toolbar, added “Capture and Save”, which just saves the captured snapshot without automatically analyzing it. It was suggested [here](#).

Alpha 1.1 (May 2nd, 2018)

- In Heap Explorer toolbar, added “Settings > Use Multi-Threading”. Previously Heap Explorer was using multi threading always. However, for debugging reasons it makes sense to provide an option to turn it off, so Unity can properly log errors to the Console window.