Finding Offsets – Object List Pointer

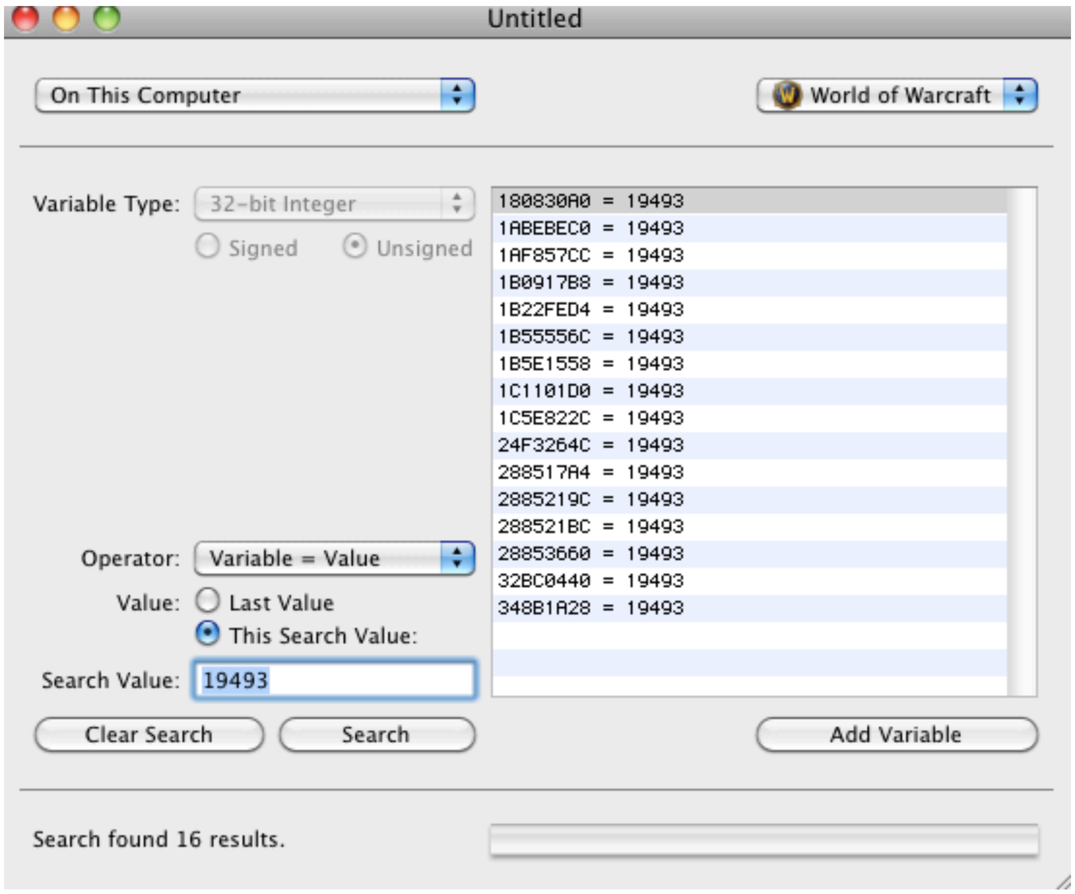
11Dec09

This is meant to be a 101 for finding offsets in memory! This is probably the most complicated address I have to find for Pocket Gnome, so if you can do this – you should be able to find all the others too 😃 Here are the requirements:

- Memory Scanner: The Cheat or iHaxGamez (Screenshots will be from The Cheat)
- Calculator (Applications -> Calculator)
- World of Warcraft
- Pocket Gnome (Memory Viewer Tab)

Step 1: Find your own object in the list!

Before we can find the object list, we first need to find our own object from within memory. So lets first search for your health! Fire up The Cheat and search for your maximum health! (I'd recommend having 100% health before you search). I searched for my health of 19493 and came up with the following results (note: the addresses are on the left, and will be different on your search):



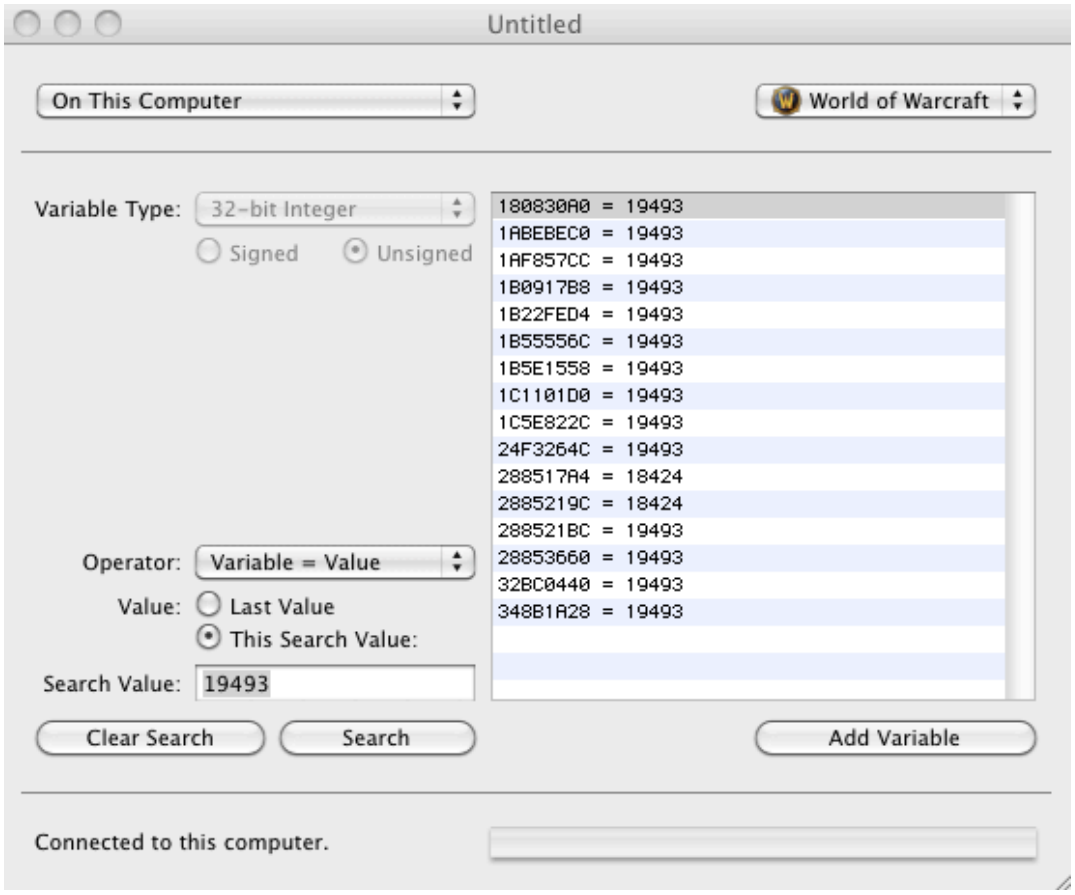
Variable Type: 32-bit Integer
Signed [✓ Unsigned]
Operator: Variable = Value
Value: Last Value [✓ This Search Value:]
Search Value: 19493

```
180830A0 = 19493
1ABEBEC0 = 19493
1AF857CC = 19493
1B0917B8 = 19493
1B22FED4 = 19493
1B55556C = 19493
1B5E1558 = 19493
1C1101D0 = 19493
1C5E822C = 19493
24F3264C = 19493
288517A4 = 19493
2885219C = 19493
288521BC = 19493
28853660 = 19493
32BC0440 = 19493
348B1A28 = 19493
```

Search found 16 results.

Health Search

Now we want to take off a piece of gear (so your health is lower), then put it back on. This will then allow us to find our current health, as it will slowly be increasing. See below at 0x288517A4 and 0x2885219C.



Variable Type: 32-bit Integer
Signed [✓ Unsigned]
Search Value: 19493
Connected to this computer.

```
180830A0 = 19493
1ABEBEC0 = 19493
1AF857CC = 19493
1B0917B8 = 19493
1B22FED4 = 19493
1B55556C = 19493
1B5E1558 = 19493
1C1101D0 = 19493
1C5E822C = 19493
24F3264C = 19493
288517A4 = 18424
2885219C = 18424
288521BC = 19493
28853660 = 19493
32BC0440 = 19493
348B1A28 = 19493
```

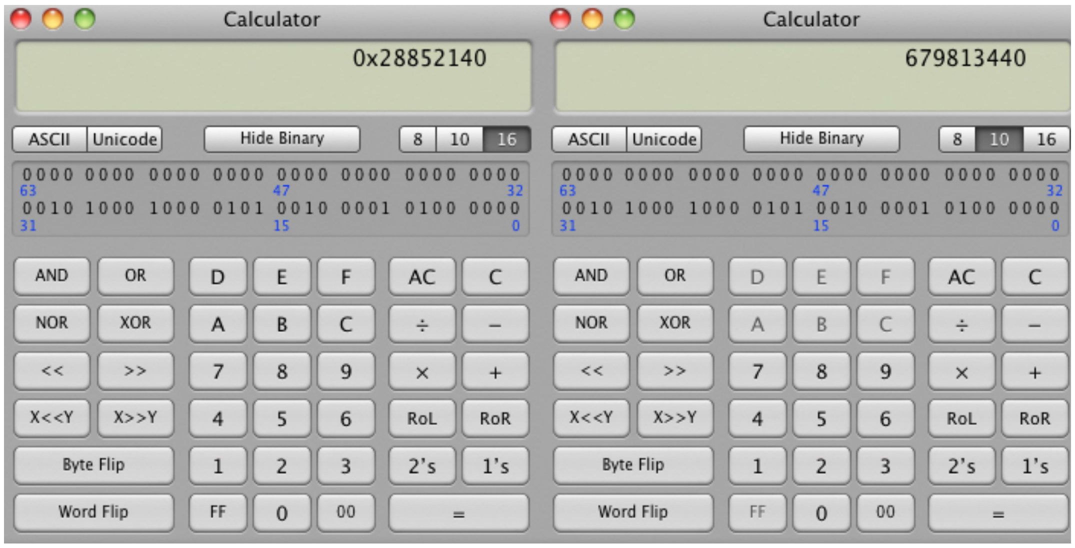
Current Health Increasing

Step 2: Find the start of our object's unitFields

The unit fields contain data such as your health, mana, rage, etc… How do we get to the "base" (or start) of our unit fields? We want to subtract 0x5C from our current health address, so:0x2885219C- 0x5C. If you don't know how to do this math, I recommend starting up Calculator, then choosing View -> Programmer. The answer is:0×28852140. What's interesting is our player's GUID (Global User ID – this is unique per object in the game) is stored at this address, it's 64-bit (clearly I won't share mine 😉).

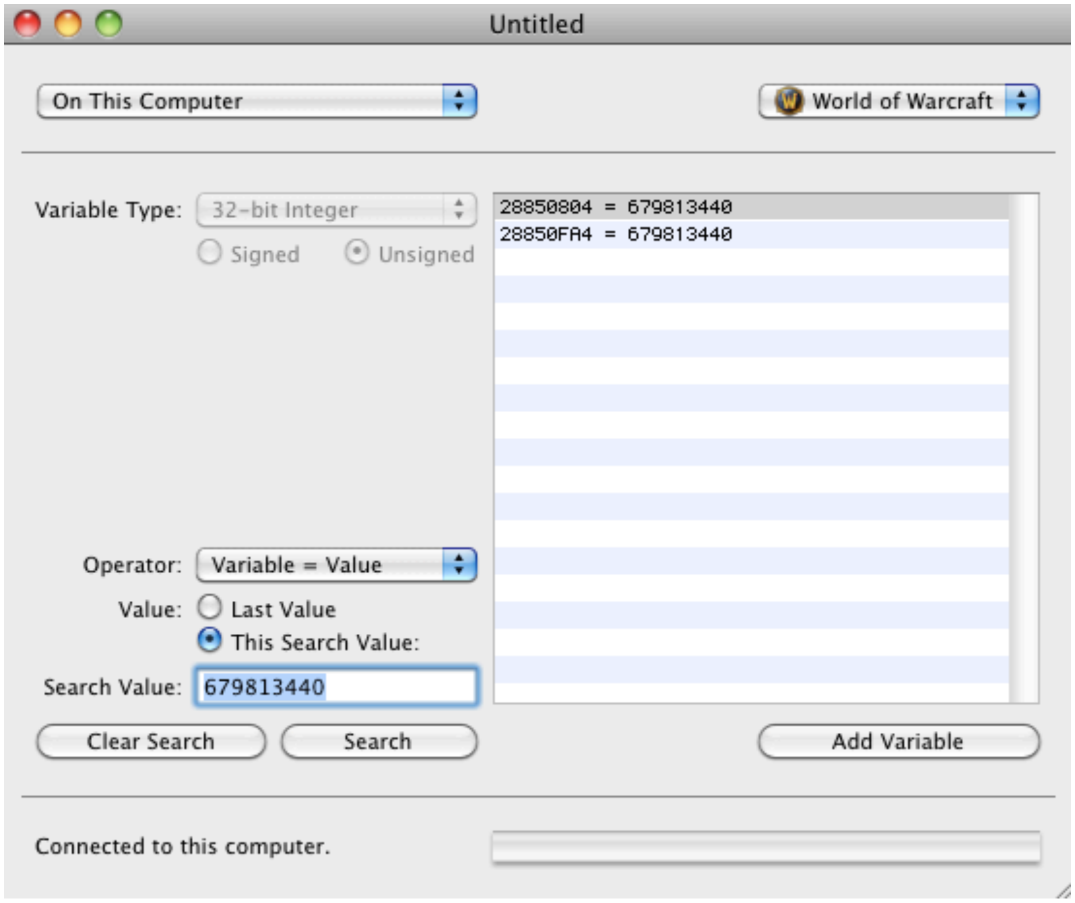
Step 3: Now find our object's base address!

Now that we have our object's unitField address (0×28852140) we can now find out object's base address! So use Calculator to convert your unitField address to a base 10.



Calculator Base 16 to Base 10

Mine is 679813440. Now we want to search for this in The Cheat (alternatively, you can right click on the memory row in Pocket Gnome and click "Find Pointers to here", this does the same thing – usually takes around 4 seconds to search).



Variable Type: 32-bit Integer
Signed [✓ Unsigned]
Operator: Variable = Value
Value: Last Value [✓ This Search Value:]
Search Value: 679813440

```
28850804 = 679813440
28850FA4 = 679813440
```

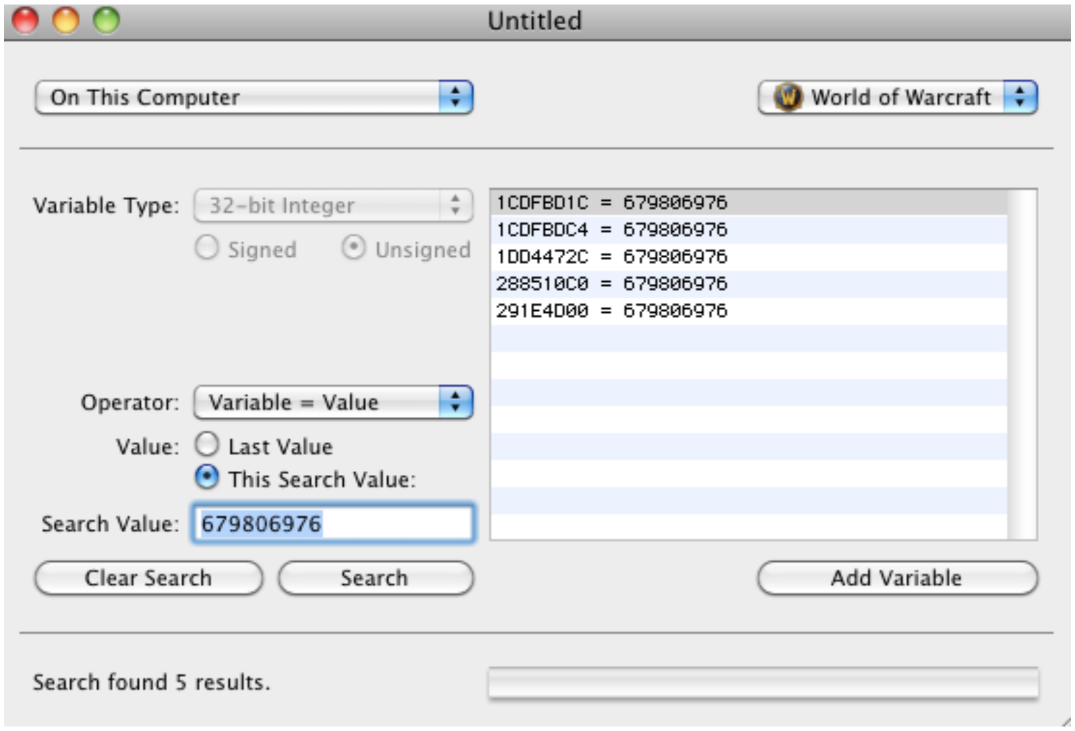
Connected to this computer.

Player Object Search

So now we have 2 options , one of these is right next to our object base! I'm going to give you a secret, it's the first one! We know this as generally objects end with 00 instead of an uneven value. This is NOT always true, nor has it been confirmed. I just know this from looking for a while. Now, 0x28850804 isn't the actual object base, rather it's 0x4 more than the object base 😃 So move back one 32-bit memory location (in other words, subtract 0×4). So 0×28850804 – 0×4 = 0×28850800. We now have our object base!

Step 4: Find the object list!

Now that we have our object base, 0×28850800, lets search in memory for where this appears. Convert your object base address to Base 10 as you did earlier using calculator. Mine is 679806976. So now I'll search for this in memory using The Cheat (or by "Find pointers to here" in Pocket Gnome). I was lucky and only had 5 values to check, your mileage may vary:



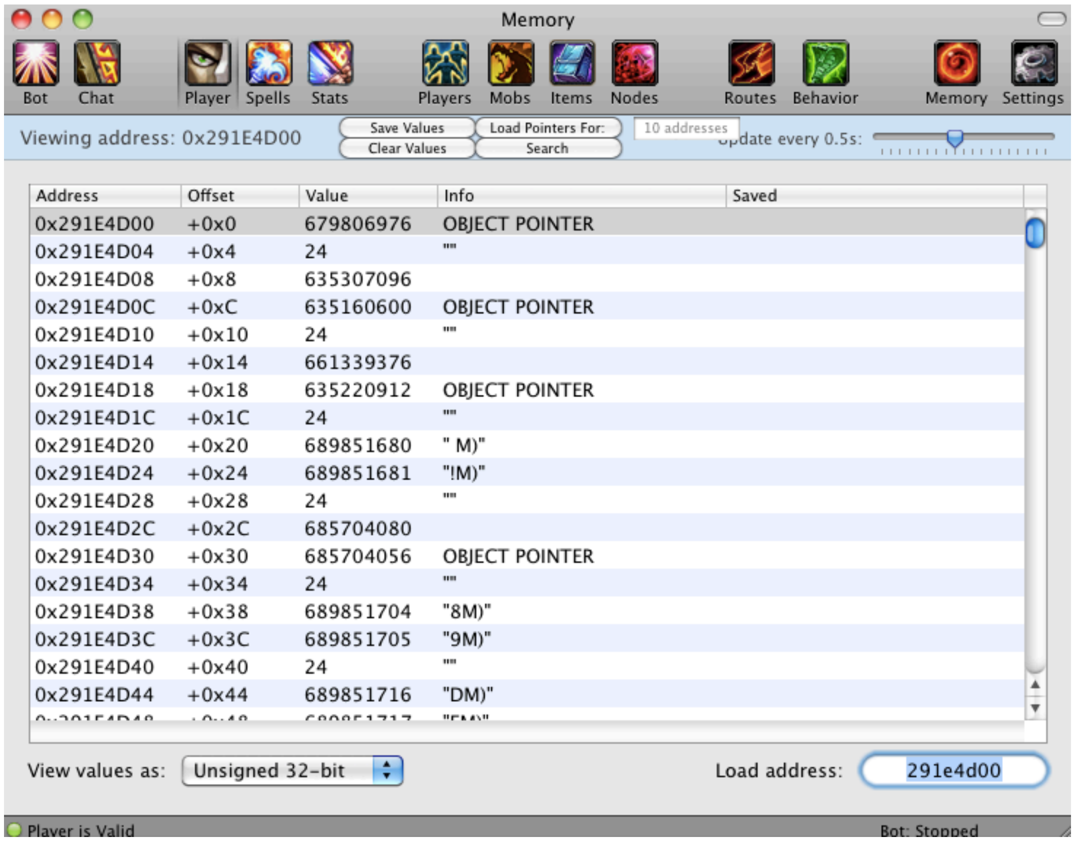
Variable Type: 32-bit Integer
○ Signed [✓ Unsigned]
Operator: Variable = Value
Value: ○ Last Value [✓ This Search Value:]
Search Value: 679806976

```
1CDFBD1C = 679806976
1CDFBDC4 = 679806976
1DD4472C = 679806976
288510C0 = 679806976
291E4D00 = 679806976
```

Search found 5 results.

Pointer to My Object

Now I'm going to type each address into Pocket Gnome's memory viewer until I find something that looks like a list (you will know as every 3rd row will be 0×18 – make sure you are viewing values as Hex 32-bit or it will appear as 24 🙂). I found that my list was at 291e4d00:



Viewing address: 0x291E4D00

Update every 0.5s:

Address	Offset	Value	Info	Saved
0x291E4D00	+0x0	679806976	OBJECT POINTER	
0x291E4D04	+0x4	24	""	
0x291E4D08	+0x8	635307096		
0x291E4D0C	+0xC	635160600	OBJECT POINTER	
0x291E4D10	+0x10	24	""	
0x291E4D14	+0x14	661339376		
0x291E4D18	+0x18	635220912	OBJECT POINTER	
0x291E4D1C	+0x1C	24	""	
0x291E4D20	+0x20	689851680	" M)"	
0x291E4D24	+0x24	689851681	"!M)"	
0x291E4D28	+0x28	24	""	
0x291E4D2C	+0x2C	685704080		
0x291E4D30	+0x30	685704056	OBJECT POINTER	
0x291E4D34	+0x34	24	""	
0x291E4D38	+0x38	689851704	"8M)"	
0x291E4D3C	+0x3C	689851705	"9M)"	
0x291E4D40	+0x40	24	""	
0x291E4D44	+0x44	689851716	"DM)"	

View values as: Unsigned 32-bit

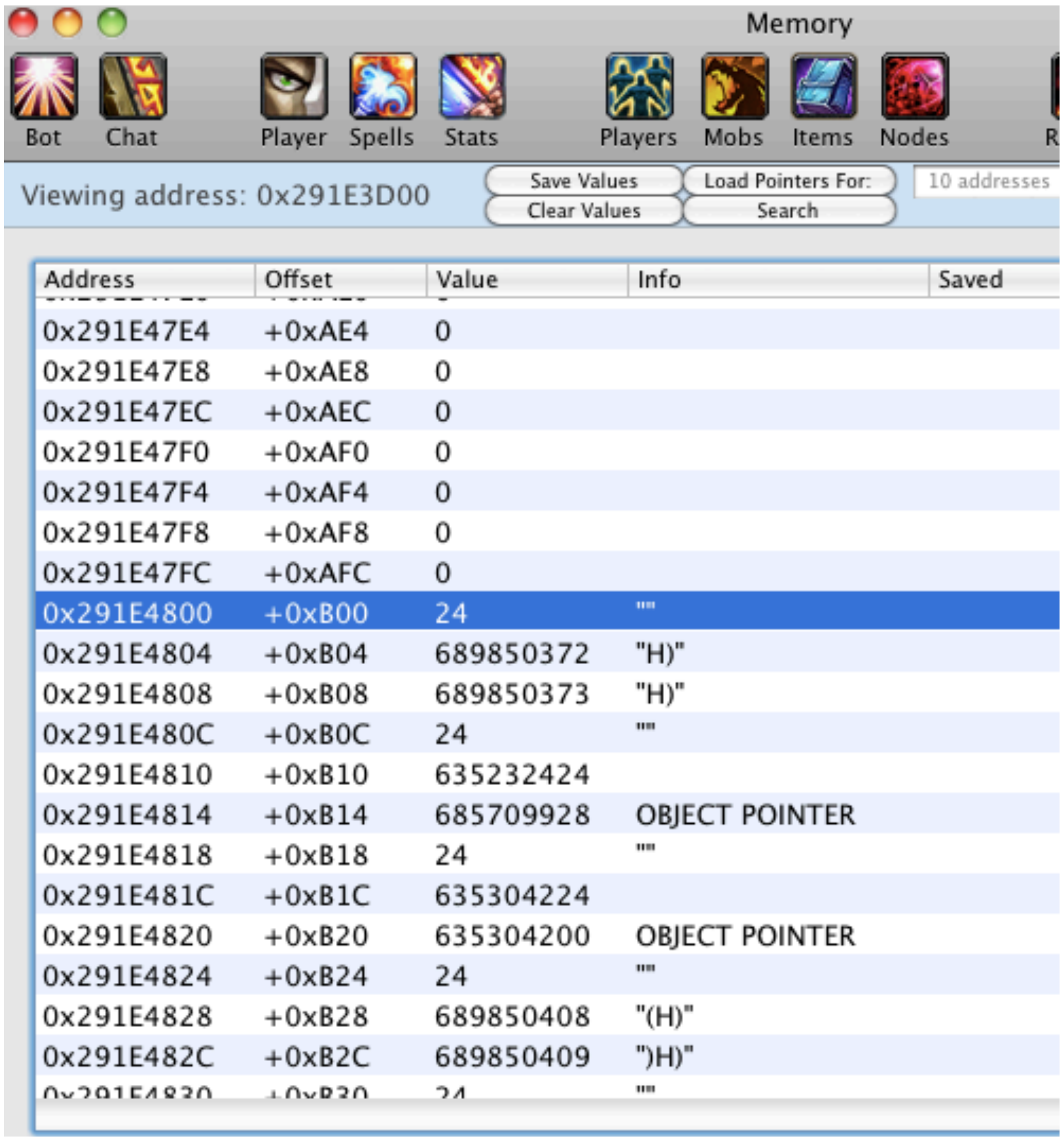
Load address: 291e4d00

Player is Valid

Bot: Stopped

Object List

Now lets jump back some until we get to the START of the list. We'll know we're at the start of the list as it will start with 0×18. I just jumped back by 0×1000, then scrolled down until I found it. So I did 0x291E3D00 – 0×1000 = 0x291E2D00 and then viewed the resulting address in Pocket Gnome. I found the start of my list is here: 0x291E4800. Wooohoo!



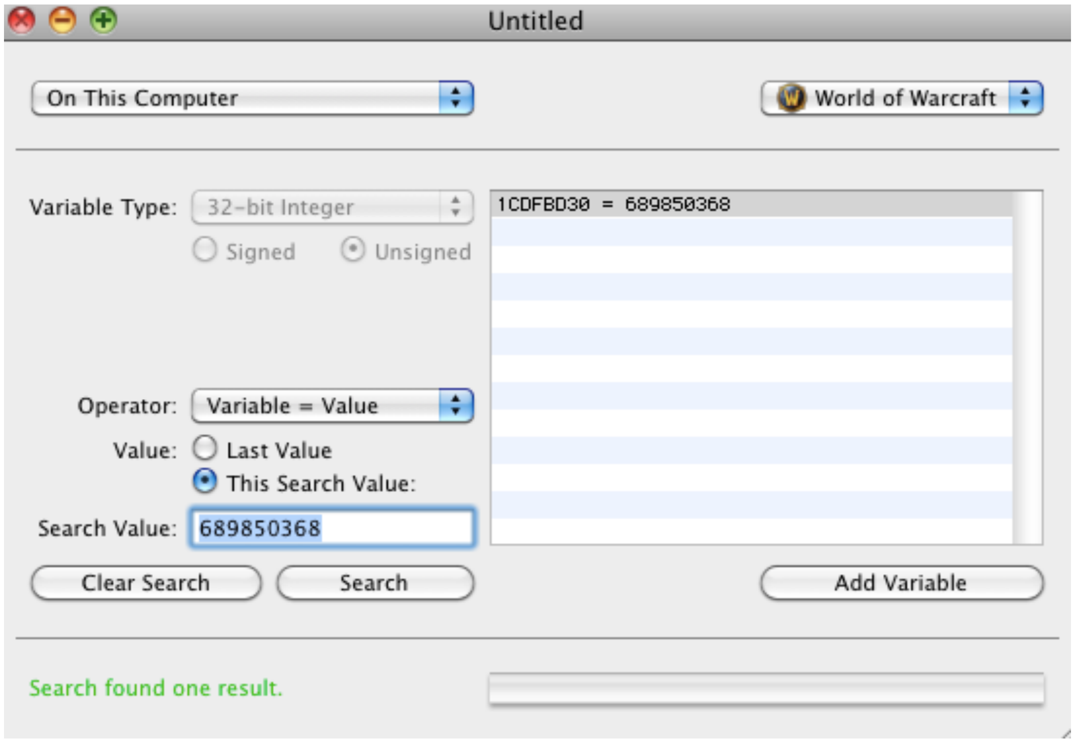
Viewing address: 0x291E3D00

Address	Offset	Value	Info	Saved
0x291E47E4	+0xAE4	0		
0x291E47E8	+0xAE8	0		
0x291E47EC	+0xAEC	0		
0x291E47F0	+0xAF0	0		
0x291E47F4	+0xAF4	0		
0x291E47F8	+0xAF8	0		
0x291E47FC	+0xAFC	0		
0x291E4800	+0xB00	24	""	
0x291E4804	+0xB04	689850372	"H)"	
0x291E4808	+0xB08	689850373	"H)"	
0x291E480C	+0xB0C	24	""	
0x291E4810	+0xB10	635232424		
0x291E4814	+0xB14	685709928	OBJECT POINTER	
0x291E4818	+0xB18	24	""	
0x291E481C	+0xB1C	635304224		
0x291E4820	+0xB20	635304200	OBJECT POINTER	
0x291E4824	+0xB24	24	""	
0x291E4828	+0xB28	689850408	"(H)"	
0x291E482C	+0xB2C	689850409	")H)"	
0x291E4830	+0xB30	24	""	

Object List Scrolled

Step 5: Find the Pointer to the Object List

Convert your object list start pointer from Base 16 to Base 10. I converted 0x291E4800 and came up with 689850368. Now lets search in the cheat (or pocket gnome) to find what points here.



Variable Type: 32-bit Integer
○ Signed ● Unsigned
Operator: Variable = Value
Value: ○ Last Value ● This Search Value:
Search Value: 689850368

```
1CDFBD30 = 689850368
```

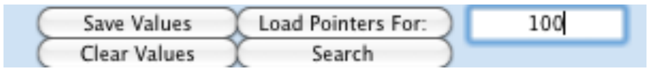
Search found one result.

Object List Static

Yay it’s one value! Mine is 0x1CDFBD30! We’re ALMOST DONE!

Step 6: Find the Static Pointer to the above address

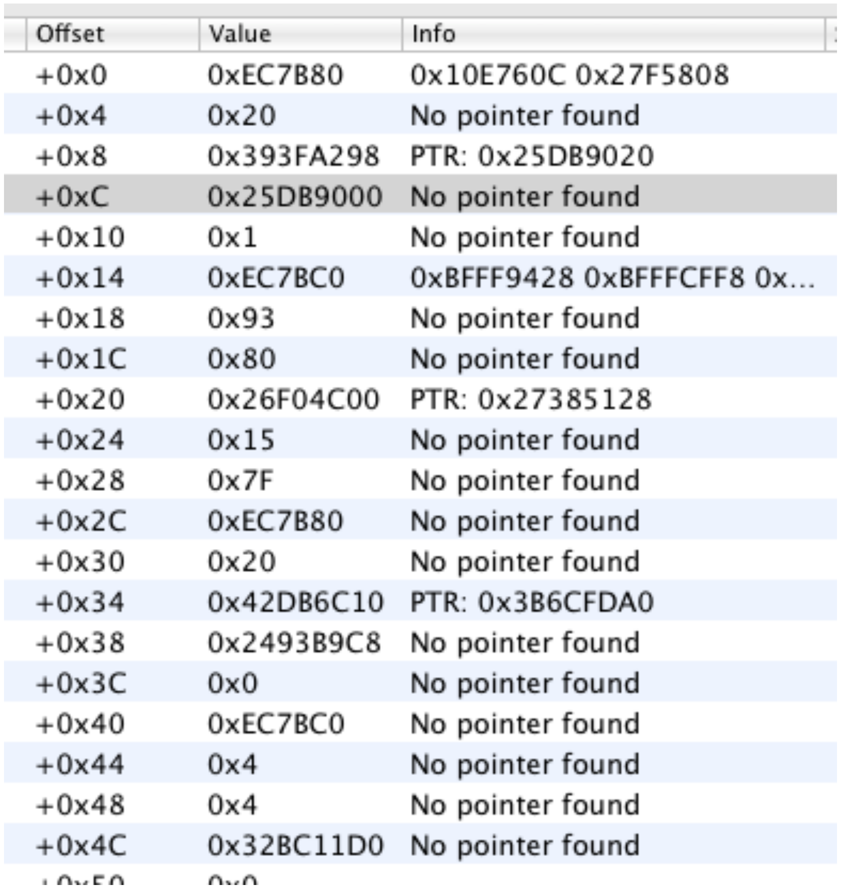
Unfortunately this part isn’t as simple as searching for 0x1CDFBD30 in The Cheat. Try this, and you will get mixed results (probably nothing). This is when I recommend you use the “Find pointers to here” feature on the Pocket Gnome Memory tab. This is MUCH easier then have to type in numbers into The Cheat. So, nothing conclusive will point to this address, but maybe something a little before this address is being pointed to by a static address! Lets subtract 0×100 from the dynamic address you received, mine is: 0x1CDFBD30 – 0×100 = 0x1CDFBC30. Now, you can right click the next 100 rows, or you can use the “scan multiple rows” feature of Pocket Gnome. To use this, simply type in 100 into the “10 addresses” box then click “Load Pointers For:”. By searching for that many values, this could take up to 10 minutes (it’s not optimized).



100

Load Pointers

Hopefully you will see one of these values being 0x10E760C – assuming you’re searching in 3.3.0. That is the static pointer WOOOHOOOO! We know this is the static pointer, because it’s around or under 0×1500000. Values above this are in a different data segment in memory + are allocated on each memory load. If you restart WoW, you will find that 0x10E760C still contains our pointer to the below 😃 And you’ll notice if you add 0×20 to that address, we’re back to our object list!



Offset	Value	Info
+0x0	0xEC7B80	0x10E760C 0x27F5808
+0x4	0x20	No pointer found
+0x8	0x393FA298	PTR: 0x25DB9020
+0xC	0x25DB9000	No pointer found
+0x10	0x1	No pointer found
+0x14	0xEC7BC0	0xBFFF9428 0xBFFFCFF8 0x...
+0x18	0x93	No pointer found
+0x1C	0x80	No pointer found
+0x20	0x26F04C00	PTR: 0x27385128
+0x24	0x15	No pointer found
+0x28	0x7F	No pointer found
+0x2C	0xEC7B80	No pointer found
+0x30	0x20	No pointer found
+0x34	0x42DB6C10	PTR: 0x3B6CFDA0
+0x38	0x2493B9C8	No pointer found
+0x3C	0x0	No pointer found
+0x40	0xEC7BC0	No pointer found
+0x44	0x4	No pointer found
+0x48	0x4	No pointer found
+0x4C	0x32BC11D0	No pointer found
+0x50	0x0	

Static Pointer

[[0x10E760C] + 0×20] = Object List Pointer

[] Represent a memory read. So First we read the value in 0x10E760C, then we add 0×20 to that value and that result is the address of our object list.

Step 7: Additional Notes

So we did just find our object list pointer, but interestingly enough Pocket Gnome doesn't use this. Through testing (figuring out the total number of objects through both methods) I was able to determine that: [[0x10E760C] + 0xB4] happens to b e a pointer to the first object in the linked list! Then we can just move through the linked list by reading [object base address + 0x34]. That is the pointer to the next object in the list. We know we're at the end by checking to see if (address&1) is true or the result read is 0. Check out Controller.m in Pocket Gnome if you'd like to see an implementation.

Let me know if you have any questions in the comments!

Filed under Uncategorized | Tagged Memory, Memory Scanning, Offsets, Searching, The Cheat, world of warcraft, wow

2 Responses to "Finding Offsets – Object List Pointer"

1. *alex321* says:
   December 12, 2009 at 6:32 pm

0×291E3D00 – 0×1000 = 0×291E3D00

typo >> = 0x291E2D00

Log in to Reply

- *tanaris4* says:
  December 12, 2009 at 7:06 pm

  Nice catch – updated

  Log in to Reply

Leave a Reply

You must be logged in to post a comment.

- Search for: Search

- **Blogroll**
  - Cypher's Blog
  - Kynox's Blog
  - Shynd's Blog

- **resources**
  - Game Deception
  - MMOwned – Memory Editing
  - Pocket Gnome
  - Reverse Engineering

- **Tags**

  byte signatures functions i386 ida injection inject_bundle intel lua mac mach mach_override Memory Memory Scanning offset Offsets os x pointers powerpc ppc reversing Searching The Cheat warden world of warcraft wow

- **Recent Entries**
  - Updating Pocket Gnome come patch day – Player Tab11.17
  - Dumping Descriptors for OS X10.4
  - Function Offsets – 4.0.3 (Build 13117)10.4
  - Warden on OS X – What I know7.7

- Lua Function Offsets – 3.3.5a (Build 12340)6.30
- Lua Function Offsets – 3.3.5 (Build 12213)6.28
- Warden – Saving the streamed modules6.25
- Lua Function Offsets – 3.3.3a (Build 11723)4.6
- inject_bundle – Issue with mach_override4.5
- Finding Offsets – Object List Pointer12.11