



Cheat Engine & RPCS3

Modding EXVSFB on RPCS3

Basic Cheat Engine setup on RPCS3

24th Aug

1

descatal

2019

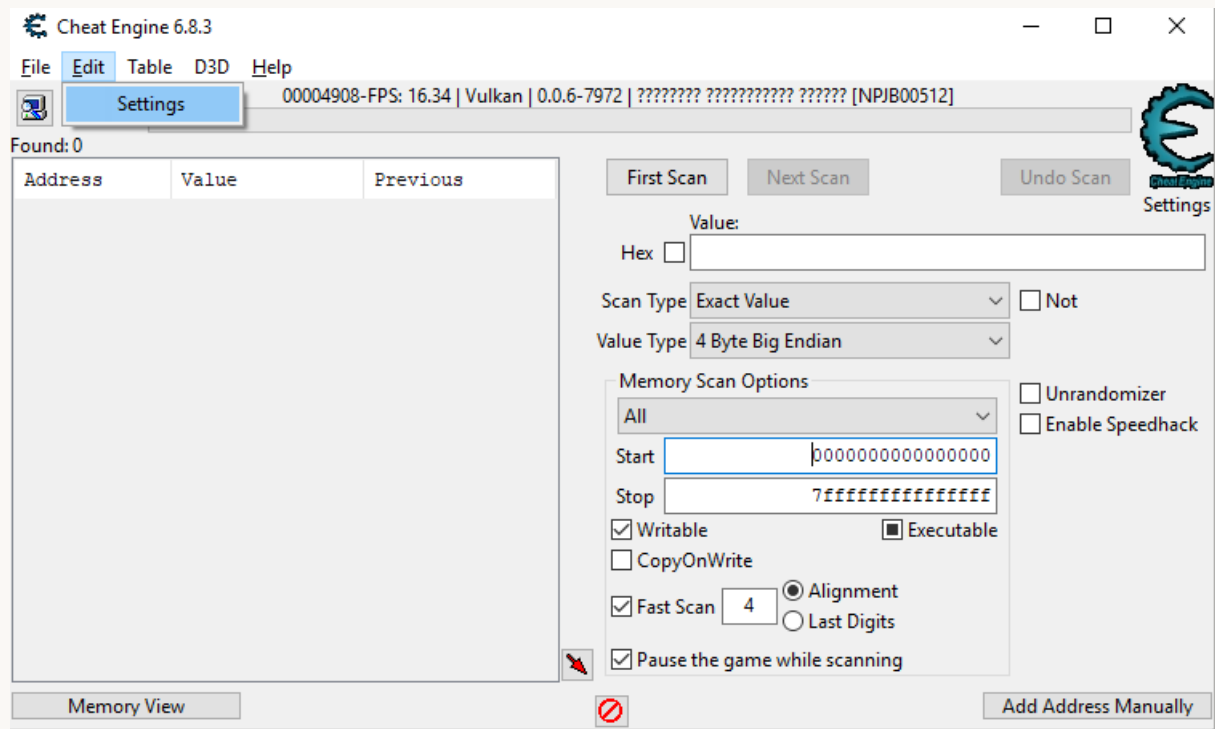
Comment

To use Cheat Engine on RPCS3, you need to set up a few things beforehand.

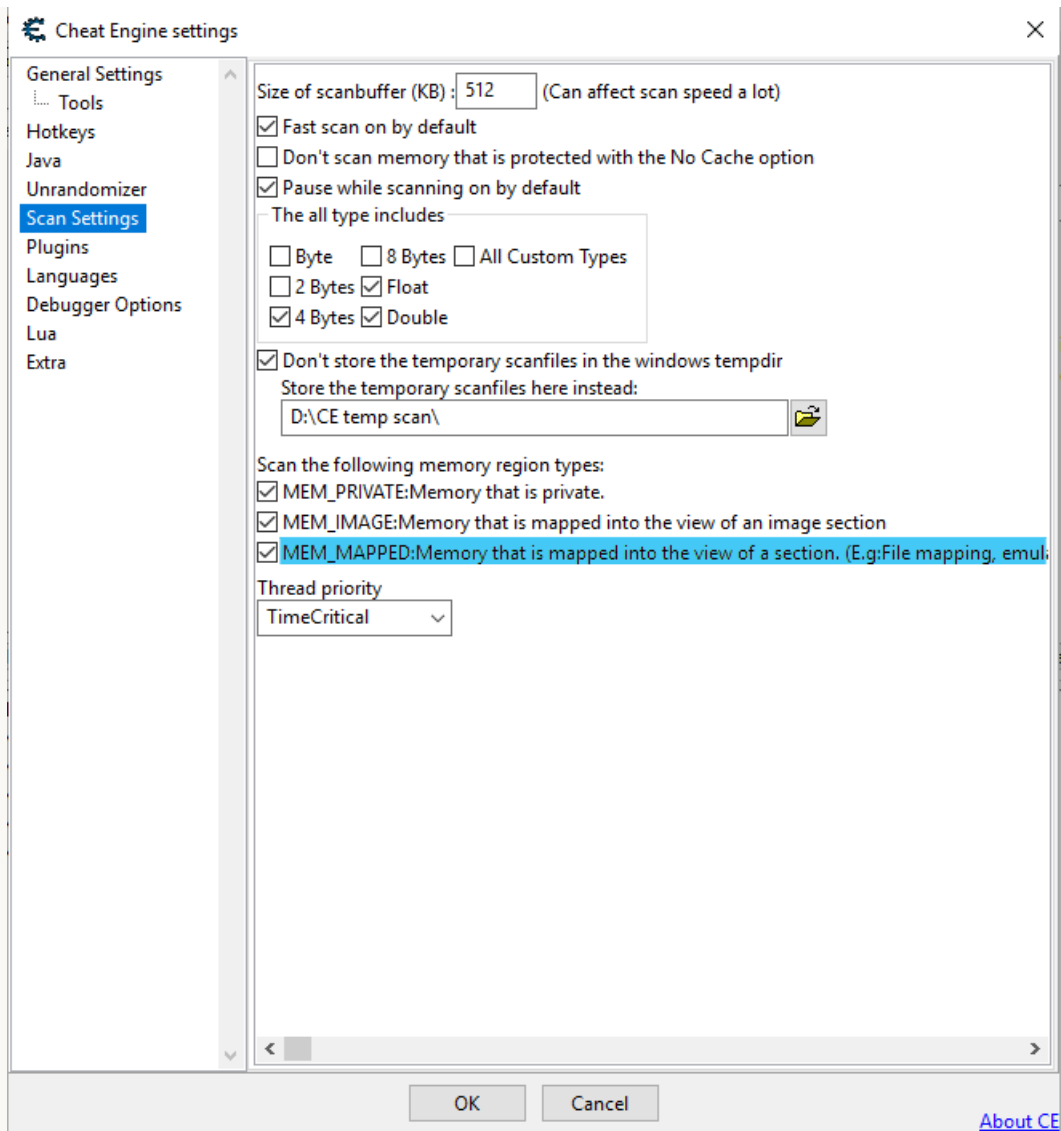
MEM_MAPPED Settings

First, we need to enable MEM_MAPPED option to let Cheat Engine have the ability to scan Mapped memory regions. As most of the games emulated are stored in mapped regions, this option is essential for using Cheat Engine in emulators.

In Cheat Engine, go to Edit -> Settings.



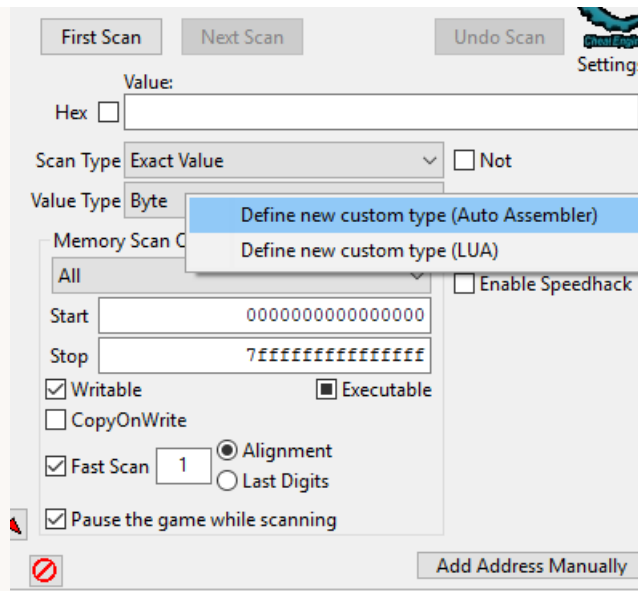
Navigate to the Scan Settings tab, and ensure that all three MEM options are enabled, as shown in the figure below.



Big Endian Support

Most emulators operates in Big Endian value type. To put it simply, Big Endian is the reversal of a normal byte array. For example, 00 01 02 03 in Big Endian will be 03 02 01 00. By default cheat engine does not include Big Endian value type, so we will need to define it ourselves.

In cheat engine, right click on the value type and select Define new custom type (Auto Assembler).



A new window will pop up with codes in it. Delete it all and paste in the codes down below.

```
alloc(TypeName,256)
alloc(ByteSize,4)
alloc(ConvertRoutine,1024)
alloc(ConvertBackRoutine,1024)
alloc(UsesFloat,1)
alloc(CallMethod,1)
```

TypeName:

```
db '2 Byte Big Endian',0
```

ByteSize:

```
dd 2
```

```
//The convert routine should hold a routine that converts the data to an
integer (in eax)
```

```
//function declared as: stdcall int ConvertRoutine(unsigned char *input);
```

```

//function declared as: static int ConvertRoutine(unsigned char input),
//Note: Keep in mind that this routine can be called by multiple threads at
the same time.
ConvertRoutine:
//jmp dllname.functionname
[64-bit]
//or manual:
//parameters: (64-bit)
//rcx=address of input
xor eax,eax
mov ax,[rcx] //eax now contains the bytes 'input' pointed to
xchg ah,al //convert to big endian

ret
[/64-bit]

[32-bit]
//jmp dllname.functionname
//or manual:
//parameters: (32-bit)
push ebp
mov ebp,esp
//[ebp+8]=input
//example:
mov eax,[ebp+8] //place the address that contains the bytes into eax
mov ax,[eax] //place the bytes into eax so it's handled as a normal 4
byte value
and eax,ffff //cleanup
xchg ah,al //convert to big endian

pop ebp
ret 4
[/32-bit]

```

//The convert back routine should hold a routine that converts the given integer back to a row of bytes (e.g when the user wats to write a new value)

//function declared as: stdcall void ConvertBackRoutine(int i, unsigned char *output);

ConvertBackRoutine:

//jmp dllname.functionname

//or manual:

[64-bit]

//parameters: (64-bit)

//ecx=input

//rdx=address of output

//example:

xchg ch,cl //convert the little endian input into a big endian input

mov [rdx],cx //place the integer the 4 bytes pointed to by rdx

ret

[/64-bit]

[32-bit]

//parameters: (32-bit)

push ebp

mov ebp,esp

//[ebp+8]=input

//[ebp+c]=address of output

//example:

push eax

push ebx

mov eax,[ebp+8] //load the value into eax

mov ebx,[ebp+c] //load the address into ebx

//convert the value to big endian

xchg ah,al

```
mov [ebx],ax //write the value into the address
```

```
pop ebx
```

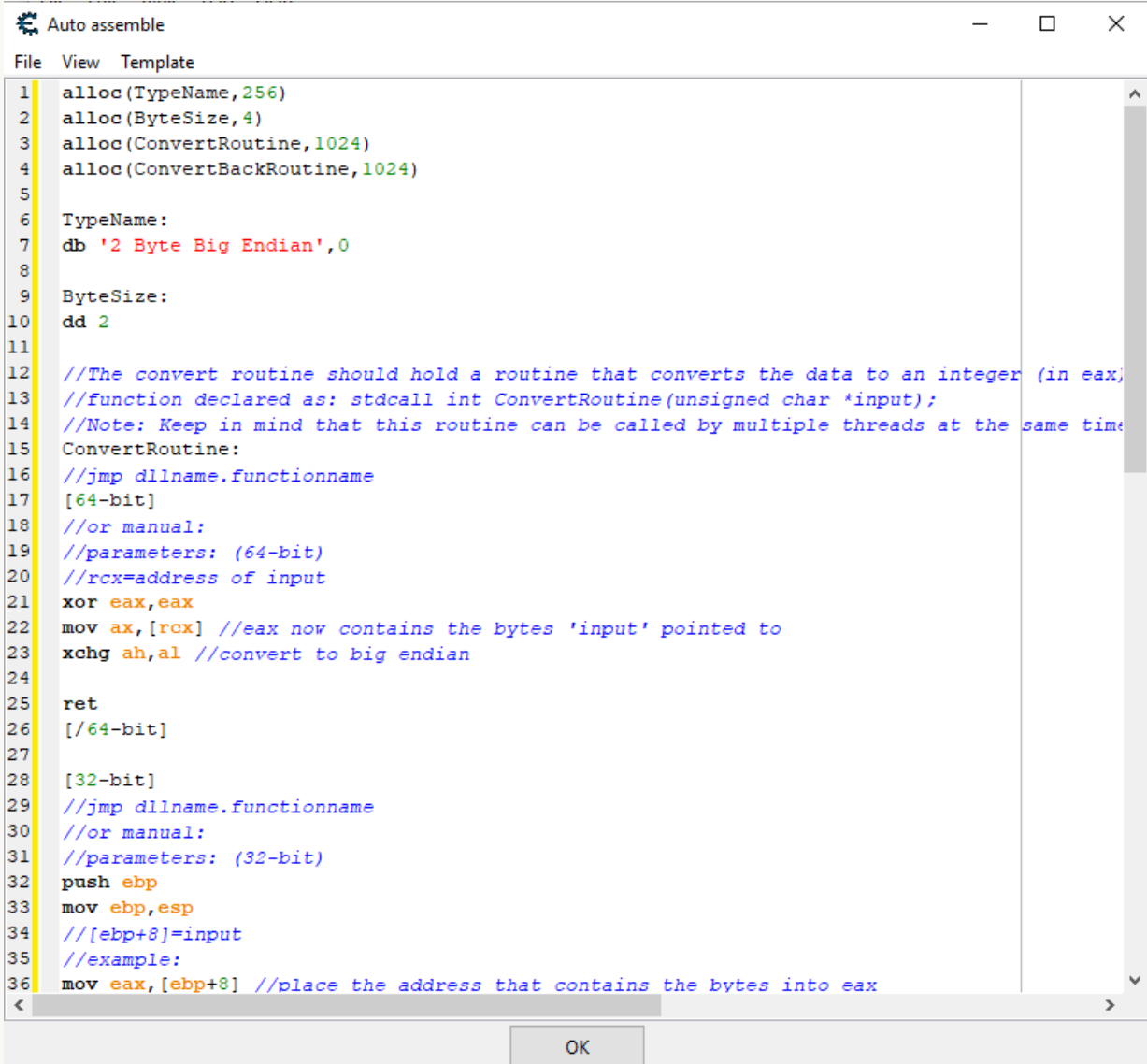
```
pop eax
```

```
pop ebp
```

```
ret 8
```

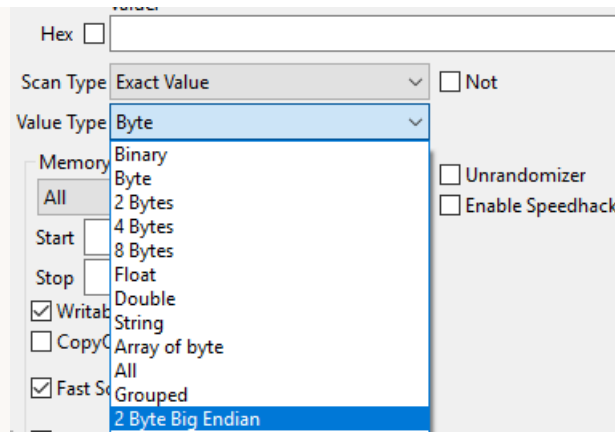
```
[/32-bit]
```

Once you have pasted it in, double check the script, and press OK



```
Auto assemble
File View Template
1  alloc(TypeName,256)
2  alloc(ByteSize,4)
3  alloc(ConvertRoutine,1024)
4  alloc(ConvertBackRoutine,1024)
5
6  TypeName:
7  db '2 Byte Big Endian',0
8
9  ByteSize:
10 dd 2
11
12 //The convert routine should hold a routine that converts the data to an integer (in eax)
13 //function declared as: stdcall int ConvertRoutine(unsigned char *input);
14 //Note: Keep in mind that this routine can be called by multiple threads at the same time
15 ConvertRoutine:
16 //jmp dllname.functionname
17 [64-bit]
18 //or manual:
19 //parameters: (64-bit)
20 //rcx=address of input
21 xor eax,eax
22 mov ax,[rcx] //eax now contains the bytes 'input' pointed to
23 xchg ah,al //convert to big endian
24
25 ret
26 [/64-bit]
27
28 [32-bit]
29 //jmp dllname.functionname
30 //or manual:
31 //parameters: (32-bit)
32 push ebp
33 mov ebp,esp
34 //[ebp+8]=input
35 //example:
36 mov eax,[ebp+8] //place the address that contains the bytes into eax
OK
```

After that, the value type you defined will be added in the value type selection, as shown below.



Using the same method, add in the 4 byte Big Endian value type.

```

alloc(TypeName,256)
alloc(ByteSize,4)
alloc(ConvertRoutine,1024)
alloc(ConvertBackRoutine,1024)
alloc(UsesFloat,1)
alloc(CallMethod,1)

```

TypeName:

```
db '4 Byte Big Endian',0
```

ByteSize:

```
dd 4
```

//The convert routine should hold a routine that converts the data to an integer (in eax)

//function declared as: stdcall int ConvertRoutine(unsigned char *input);

//Note: Keep in mind that this routine can be called by multiple threads at the same time.

ConvertRoutine:

```
//jmp dllname.functionname
```

```
[64-bit]
```

//or manual:

```
//parameters: (64-bit)
```

```
//parameters: (32-bit)
```

```
//rcx=address of input
```

```
xor eax,eax
```

```
mov eax,[rcx] //eax now contains the bytes 'input' pointed to
```

```
bswap eax //convert to big endian
```

```
ret
```

```
[/64-bit]
```

```
[32-bit]
```

```
//jmp dllname.functionname
```

```
//or manual:
```

```
//parameters: (32-bit)
```

```
push ebp
```

```
mov ebp,esp
```

```
//[ebp+8]=input
```

```
//example:
```

```
mov eax,[ebp+8] //place the address that contains the bytes into eax
```

```
mov eax,[eax] //place the bytes into eax so it's handled as a normal 4  
byte value
```

```
bswap eax
```

```
pop ebp
```

```
ret 4
```

```
[/32-bit]
```

//The convert back routine should hold a routine that converts the given integer back to a row of bytes (e.g when the user wats to write a new value)

```
//function declared as: stdcall void ConvertBackRoutine(int i, unsigned  
char *output);
```

```
ConvertBackRoutine:
```

```
//jmp dllname.functionname
```


//or manual:

[64-bit]

//parameters: (64-bit)

//ecx=input

//rdx=address of output

//example:

bswap ecx //convert the little endian input into a big endian input

mov [rdx],ecx //place the integer the 4 bytes pointed to by rdx

ret

[/64-bit]

[32-bit]

//parameters: (32-bit)

push ebp

mov ebp,esp

//[ebp+8]=input

//[ebp+c]=address of output

//example:

push eax

push ebx

mov eax,[ebp+8] //load the value into eax

mov ebx,[ebp+c] //load the address into ebx

//convert the value to big endian

bswap eax

mov [ebx],eax //write the value into the address

pop ebx

pop eax

pop ebp

ret 8

[/32-bit]

And the same for Float Big Endian

```
alloc(TypeName,256)
alloc(ByteSize,4)
alloc(ConvertRoutine,1024)
alloc(ConvertBackRoutine,1024)
alloc(UsesFloat,4)
alloc(CallMethod,1)
```

TypeName:

```
db 'Float Big Endian',0
```

ByteSize:

```
dd 4
```

UsesFloat:

```
db 01
```

ConvertRoutine:

```
[32-bit]
```

```
push ebp
```

```
mov ebp,esp
```

```
mov eax,[ebp+8] //place the address that contains the bytes into eax
```

```
mov eax,[eax] //place the bytes into eax
```

```
bswap eax
```

```
pop ebp
```

```
ret 4
```

```
[/32-bit]
```

```
[64-bit]
```

```
//rcx=address of input
```

```
mov eax,[rcx] //eax now contains the bytes 'input' pointed to
```

```
bswap eax
```

```
ret
```

[/64-bit]

ConvertBackRoutine:

[32-bit]

push ebp

mov ebp,esp

//[ebp+8]=input

//[ebp+c]=address of output

push eax

push ebx

mov eax,[ebp+8] //load the value into eax

mov ebx,[ebp+c] //load the address into ebx

bswap eax

mov [ebx],eax //write the value into the address

pop ebx

pop eax

pop ebp

ret 8

[/32-bit]

[64-bit]

//ecx=input

//rdx=address of output

bswap ecx

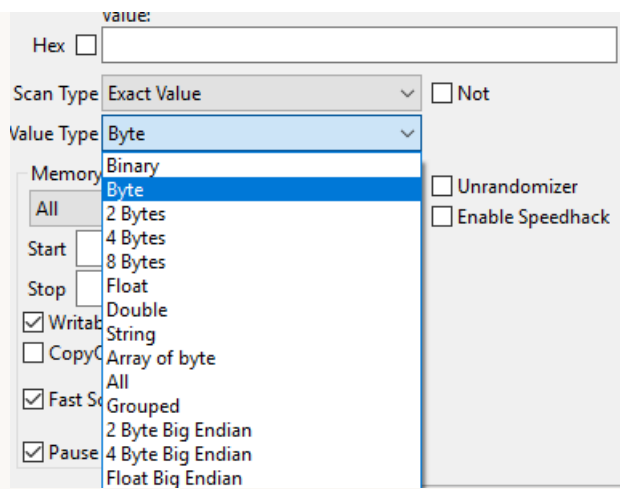
mov [rdx],ecx //place the integer the 4 bytes pointed to by rdx

ret

[/64-bit]

Once you pressed OK, you are set to go!

But just to be sure, check if your value type has 3 new selections available.



And when you want to scan for values in game, please use their big endian counterpart.

Loading...

24th Aug

descatal 2019

Uncategorized

Big Endian, Cheat Engine, 逆字节, 设置, Full Boost, Hack, PS3, RPCS3, Settings, setup

—Previous Post

How to play EXVSFB on RPCS3

Next Post—

How to modify Health, Boost and EX in EXVSFB

Join the Conversation



1 Comment

Pingback:

How to modify Health, Boost and EX in EXVSFB – Gundam Extreme VS Full Boost – Cheat Engine & RPCS3

Gundam Extreme VS Full Boost – Cheat Engine & RPCS3 Blog at WordPress.com.