

# Cyber Space Odyssey - HOWTO INSTALL

---

- Install MSVC 2022 w/ C++ Support so you can write code to send UDP packets
- Install Wireshark so you can capture the server/client traffic on the loopback interface
- Download and extract CyberSpaceOdyssey's .zip archive.
  - Launch CyberSpaceOdyssey's Server via `Launch Server.bat`
  - Launch CyberSpaceOdyssey's Client via `Launch Client.bat`
  - Play CyberSpaceOdyssey using the Client graphical window!

## Download CyberSpaceOdyssey

---

CyberSpaceOdyssey does not need to be installed, only downloaded and extracted.

1. In your web browser, navigate to <http://www.nykl.net/cso/> and download [CyberSpaceOdyssey\\_release.zip](#). You may save the archive anywhere on your filesystem. In this example, we save

CyberSpace0dyssey\_release.zip to C:\repos\myfolder.

Index of /cso/

Name	Modified	Size
Parent directory	-	-
<a href="#">doc/</a>	16-Aug-2022 15:35	[DIRECTORY]
<a href="#">CSO Trailer v1.0_small.mp4</a>	26-Jul-2022 08:38	25.4M
<a href="#">CSO Trailer v1.0.mp4</a>	25-Jul-2022 15:18	185.2M
<a href="#">CyberSpace0dyssey_release.zip</a>	16-Aug-2022 15:04	27.5M

CSO Trailer v1.0.mp4 25-Jul-2022 15:18 185.2M

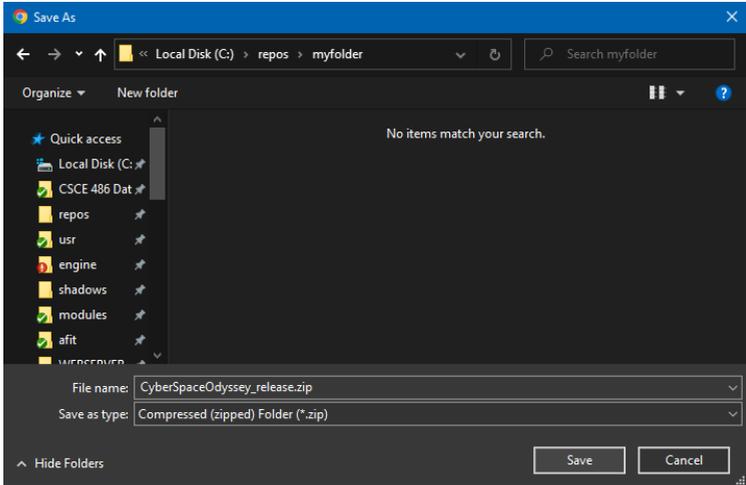
CyberSpace0dyssey\_release.zip 16-Aug-2022 15:04 27.5M

- Open link in new tab
- Open link in new window
- Open link in incognito window
- Save link as...
- Copy link address
- AdBlock — best ad blocker
- Inspect

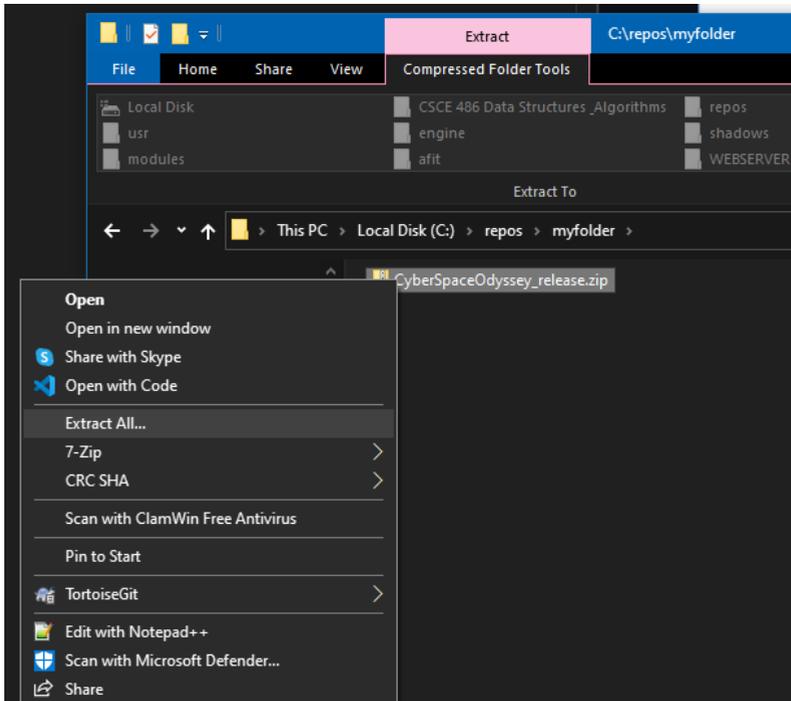
www.nykd.net/cso/CyberSpace0dyssey\_release and std::erase\_i

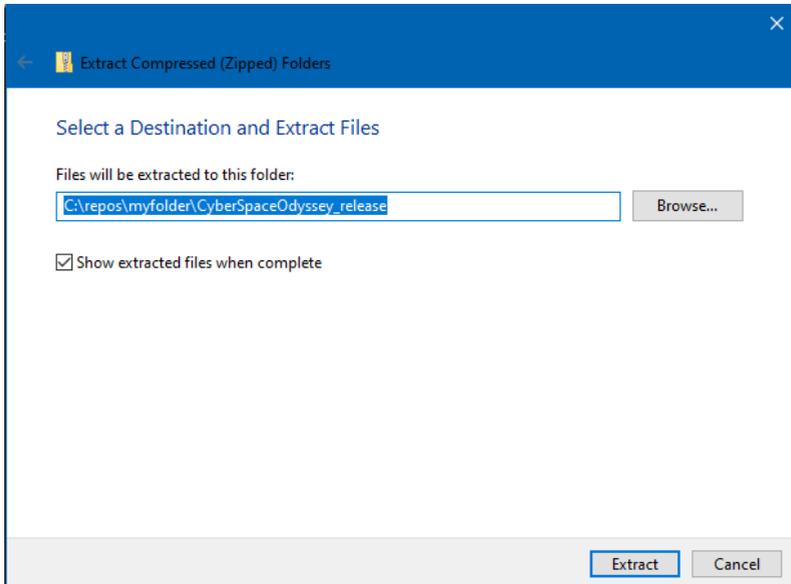
the programmer to explicitly

Limitation

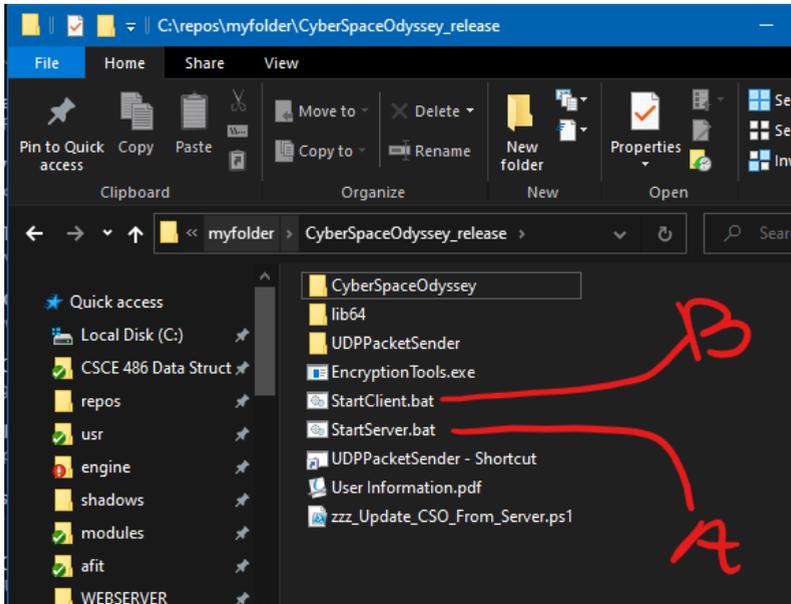


2. Next we will extract the .zip archive. Navigate to the folder where the archive was downloaded. To extract the archive, right click and selected `Extract All` . Then press `Extract` .





1. Now you can browse to the root of CyberSpaceOdyssey. The following files should be visible. The StartServer.bat (red letter A) launches the server. The StartClient.bat (red letter B) launches the client.



4. To play, one must first launch a server by double clicking `StartServer.bat` . After the server loads, you may move those windows to the side of your screen. You will not need to ever provide inputs to the server console or graphical window - it simply visualizes all the attached clients.
5. After the server is running, launch the client by double clicking `StartClient.bat` . The client is where you will spend your time playing Cyber Space Odyssey. All flying, hailing, issuing commands, etc will be done via the client's graphical window.

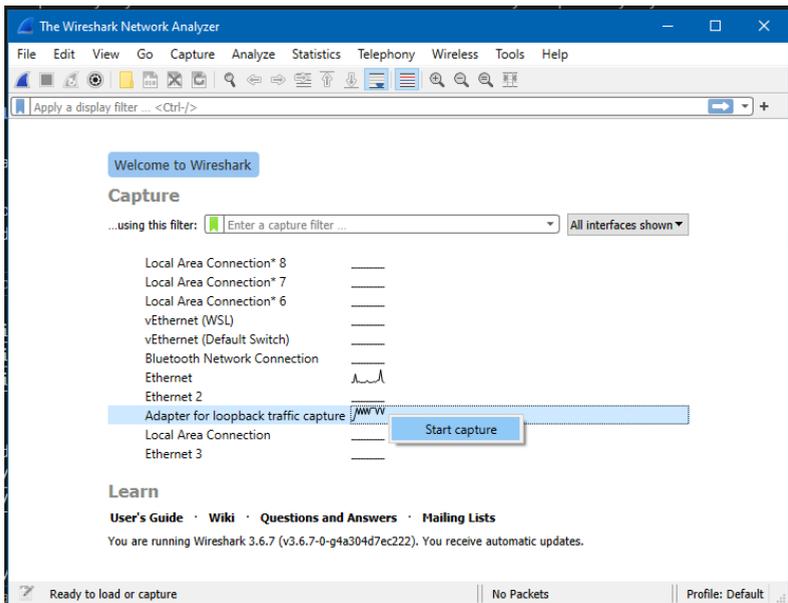
## Download Wireshark

---

In order to capture and inspect the network traffic sent between the Server and Client, you must install Wireshark on your machine. Wireshark may be downloaded and installed, for free, from <https://www.wireshark.org/download.html>. On Windows, choose the

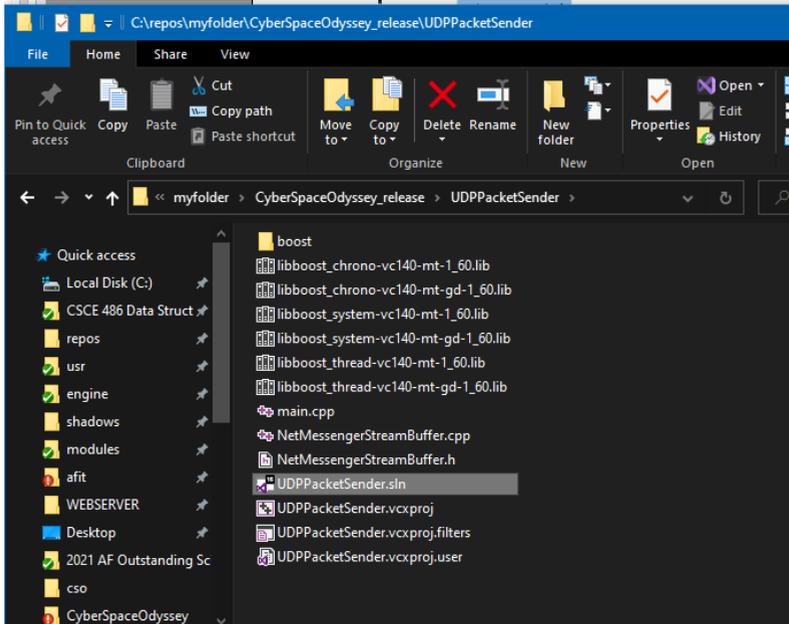
Windows Installer (64-bit). Run the installer and use all the default settings.

Once installed, you may capture packets using the *loopback interface* between the Server and Client, both of which are running on your local machine. The image below shows how to begin a capture -> Right click on Adapter for loopback traffic capture and choose Start capture .



## Crafting and Sending your own UDP Packets

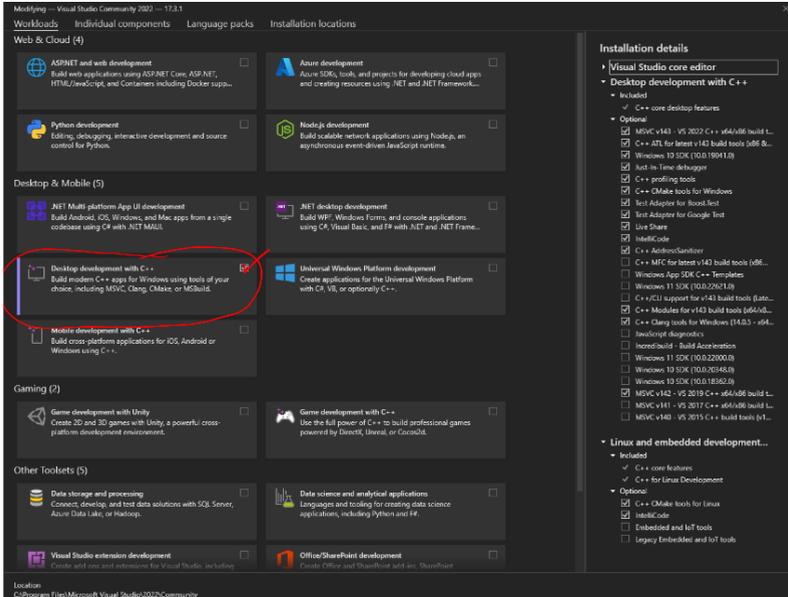
The within the root folder of your extracted `CyberSpace0dyssey_release` folder, you will see a subfolder called `UDPPacketSender` , this contains a Microsoft Visual Studio 2022 solution file named `UDPPacketSender.sln` which will open a small source code project that will let you craft custom UDP Packets to send to the server in response to events that happen within the client's game.



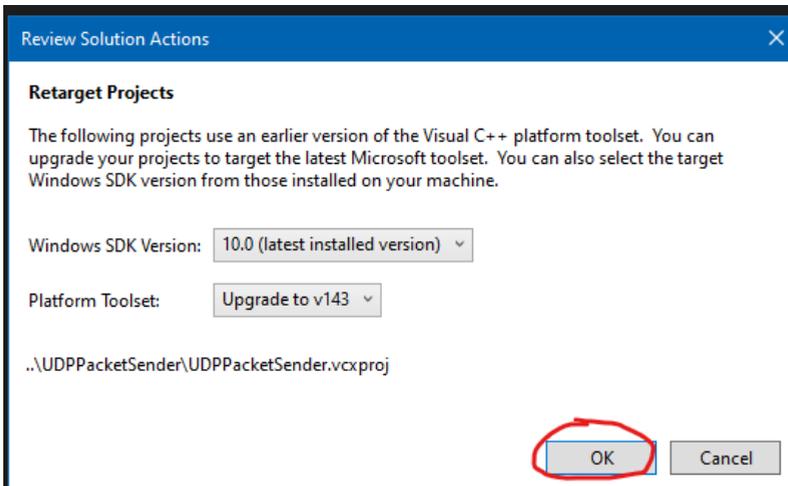
## Install Microsoft Visual Studio 2022 (Community edition is *free*)

To open the UDP Packet Sender, one must use Microsoft Visual Studio 2022 from Microsoft at <https://visualstudio.microsoft.com/vs/>.

During installation, be sure to check the `Desktop development with C++` checkbox before continuing with the normal installation - this let's you compile and run the C++ source code.



Upon double clicking the `.sln` file, MSVC 2022 may ask if you would like to upgrade to the latest SDK. Choose “Yes”



To craft a packet, change the source code between lines 185 and 192 in main.cpp. After you set the variables to their desired values, press the *green play* button to compile and run the code - which will transmit your crafted UDP Packet.

```
main.cpp x What's New?
[UDPPacketSender]
196
197 //Second I need the data payload
198
199 //Here's my header / net message number
int headerID = NetMsgCypherResponse;
200
201
202 int dummyPayloadLengthInBytes = 0; //We have to keep 4 bytes available to store they payload length immediate
203 //However, we won't know what the packet payload length will be until all
204 //Once all the data is inserted, we can call is.updatePayloadLengthInHeader
205 //the total payload length and update the 4 byte integer immediately after
206 //HMM!!!DUMB... 2000
207 //N = header ID, L = payload length, D = bytes of data.
208
209 //Use the NetMessageStreamBuffer class to create a buffer we can easily
NetMessageStreamBuffer is = NetMessageStreamBuffer( true ); //create a buffer that owns and manages its o
210
211 is << headerID; //first set header, first 4 bytes
212 is << dummyPayloadLengthInBytes; //DON'T FORGET THIS LINE!!! These next 4 bytes get updated when we call is.u
213
214
215 int clientID = 0; // who is sending the packet?
216 int stationID = 3; // who is receiving the packet?
217 int answer = 0; // what is the answer? don't forget to ensure you send an int where you should send an in
218
219 is << clientID;
220 is << stationID;
221 is << answer;
222
223 is.updatePayloadLengthInHeader();
224
225 //msg buffer is populated and we are ready to send!
226 unsigned int sz = is.getBufferLen();
227 sock_send_to( boost::asio::buffer( is.getBuffer(), is.getBufferLen() ), ep, 0, errCode );
228 std::cout << __LINE__ << " Stream is\n" << is.toStringCurrentContents() << "\n";
229
230
231
232
233 int main( int argc, char *argv[] )
234 {
235 //Initialize variables
236 boost::asio::io_service io_service; // Input/Output service to/from Operating System. We're using C++ Boost a
237 boost::asio::ip::udp::socket socket( io_service ); //Create a udp socket using the Operating System I/O decla
238 boost::asio::ip::udp::endpoint remote_endpoint; //Stores the ipid address and port number where our UDP packe
239 boost::system::error_code errCode; //When we perform a_send() operation on a socket object, this variable is
240 //We should always check errCode after each socket_send_to(...) invocation
241 socket.open( boost::asio::ip::udp::v4() ); //This "opens" the udp socket so we can begin using it to send pac
242
243
244
245 //The destination where our UDP packets are sent is defined below. We specify both the IPv4 address and port
246 remote_endpoint = boost::asio::ip::udp::endpoint( boost::asio::ip::address::from_string( "127.0.0.1" ), 12683
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

## Using NetMsgs and when to send them

How do I know what NetMsg to send when? Read pg.5 of [User Information.pdf](#) (Section 11 Summary of NetMsg Payloads) to see what NetMsgs to send in response to a particular event. See pg. 6 of [User Information.pdf](#) (Section 12 NetMsg Packet Structure) for a source code example.

[User Information.pdf](#) is located at the root of your `CyberSpace0dyssey_release` folder. An online version is also available at <http://www.nykl.net/cso/User%20Information.pdf>.