

.....
.....
DRIVING AND LISTENING TO X-PLANE
.....
.....

This document will tell you how to get data from or send data to X-Plane!
Use this data for whatever purposes you like, including running custom cockpits, motion platforms, or anything else you can think of.
I made X-Plane to be used, so you should use these messages to drive it, or listen to it, in any way you can imagine, without restriction.

This document shows how to send data to and from X-Plane by UDP messages over ethernet cable or WIFI,
but you can also Google “plugins for X-Plane” to learn how to write a plugin that actually RUNS INSIDE X-PLANE ITSELF,
giving much FASTER control of MORE X-Plane variables... but you have to learn how to write a plugin to do that.
So PLUGINS are the way to read and write lots of stuff to and from X-Plane, fast.

BUT, there is SOME stuff you can do WITHOUT plugins, by writing your OWN app to simply send and receive data by UDP over ethernet or WIFI!
Cool!

With the UDP messages described in this document, you don't have to learn plugins if you already know how to send and receive UDP messages from your own App.

.....
.....
NOTES
.....
.....

NOTE: X-Plane always receives on port 49000.

NOTE: Any strings that you send should be null-terminated!

NOTE: GO TO THE SETTINGS MENU, OPERATIONS AND WARNINGS SCREEN, AND TURN ON THE “dump net data to log.txt” button to have X-Plane dump log data to the file error.out.

This will slow down the sim a little, but you can leave it on for a little while as you get started, so that if your data transfer does NOT work, you can quit X-Plane, open the error.out file with a text editor, and see what error messages X-Plane is giving you based on the data you are sending in!

.....
.....
GET STARTED
.....
.....

OK let's get the first two sections out of the way since they are the only two things that anyone ever asks me for:
Driving X-Plane as a visual system for your own flight model, or using X-Plane's flight model to drive your own visual system.

.....
.....
JUST LET ME GET OUTPUT FROM X-PLANE TO DRIVE MY VISUAL OR
MAP: RPOS
.....
.....

Now: A UDP output from X-Plane to drive you own visuals or moving map!

Send the following data to X-Plane's IP address by UDP into port 49,000:

The 5 chars "RPOS" (four chars plus a NULL at the end!) followed by a STRING (also null-terminated of course!) that lists the number of positions per second that you want X-Plane to send right back to you to drive your visual system.

So send "RPOS_60_", where the _ is a null, to have X-Plane send you it's position 60 times per second.

NOTE: X-Plane will send the message right back to the IP address and port number you sent the RPOS command FROM!

This is in the settings menu, internet settings screen, right-most tab for data output.

And here is the data that will come to you, with no annoying struct-alignment issues: The data is all perfectly tightly packed with no spacing.

```
the four chars RPOS and a NULL.  
double dat_lon      longitude of the aircraft in X-Plane of  
course, in degrees  
double dat_lat      latitude  
double dat_ele      elevation above sea level in meters  
float y_agl_mtr     elevation above the terrain in meters  
float veh_the_loc   pitch, degrees  
float veh_psi_loc   true heading, in degrees  
float veh_phi_loc   roll, in degrees  
float vx_wrl        speed in the x, EAST, direction, in  
meters per second  
float vy_wrl        speed in the y, UP, direction, in meters  
per second  
float vz_wrl        speed in the z, SOUTH, direction, in  
meters per second  
float Prad          roll rate in radians per second  
float Qrad          pitch rate in radians per second  
float Rrad          yah rate in radians per second
```

So, this is very simple, and will let you drive your moving maps and external visuals, which is the most commonly used output from X-Plane!

```
.....  
.....  
JUST LET ME GET THE WEATHER-RADAR FROM X-PLANE: RADR  
.....  
.....
```

Now: A UDP output from X-Plane to drive you own weather radar display... perhaps for your own moving map!

Send the following data to X-Plane's IP address by UDP into port 49,000:

The 5 chars "RADR" (four chars plus a NULL at the end!) followed by a STRING (also null-terminated of course!) that lists the number of radar points per frame that you want X-Plane to send right back to you to drive your visual system.

So send "RADR_10_", where the _ is a null, to have X-Plane send you 10 radar data points per frame.

NOTE: X-Plane will send the message right back to the IP address and port number you sent the RADR command FROM!

This is in the settings menu, internet settings screen, right-most tab for data output.

And here is the data that will come to you, with no annoying struct-alignment issues: The data is all perfectly tightly packed with no spacing.

```
the four chars RADR and a NULL.  
float lon                longitude of the radar point  
float lat                latitude of course  
float storm_level_0_100  precip level, 0 to 100  
float storm_height_meters the storm tops in meters above  
sea level
```

So, this is very simple, and will let you drive your moving maps and external visuals, which is the most commonly used output from X-Plane!

.....
.....
JUST LET ME GET REAL-TIME FLIR-IMAGERY FROM X-PLANE: FLIR
.....
.....

Send the following data to X-Plane's IP address by UDP into port 49,000:

The 5 chars "FLIR" (four chars plus a NULL at the end!) followed by a STRING (also null-terminated of course!) that lists the number of FLIR frames per second that you want X-Plane to send right back to you. So send "FLIR_10_", where the _ is a null, to have X-Plane send you 10 FLIR images per second.

See the document: "Getting X-Plane real-time FLIR.rtf" for full code on how to extract the FLIR images that will be sent to you.

.....
.....

JUST LET ME USE MY FLIGHT MODEL TO DRIVE X-PLANE AS A VISUAL DISPLAY: VEHX

.....
.....

OK YOU HAVE A GREAT FLIGHT MODEL BUT A BAD VISUAL SYSTEM, so now you want to drive X-Plane as a visual system!
OR, you want to drive ALL the airplanes in X-Plane all about the sky as you like.

OK! Then send the following data to X-Plane's IP address by UDP into port 49,000:

The 5 chars "VEHX" (four chars plus a NULL at the end!) followed by the following data, with no annoying spaces in between the data due to struct alignment at all.

This is just perfectly-packed data:

```
int p                The index of the airplane you want
to control. use 0 for the main airplane that you fly to drive
the visuals.
double dat_lat       latitude, in degrees
double dat_lon       longitude, in degrees
double dat_ele       elevation above sea level, in
meters
float veh_psi_true   heading, degrees true
float veh_the        pitch, degrees
float veh_phi        roll, degrees
```

.....
.....

OTHER UDP MESSAGES TO X-PLANE

.....
.....

So those are plugins (which let you do anything) and and the VEHX, RPOS, and RADR messages to let you drive X-Plane as a visual system, or use X-Plane to drive YOUR visual system.

Now, let's go over various OTHER messages you can send to X-Plane to do various OTHER stuff!

Below, you will see some variable types that are defined internally to X-Plane, and here they are:

XCHR (character, in local byte-order for the machine you are on)
XINT (4-byte int, in local byte-order for the machine you are on)
XFLT (4-byte ints and floats, in local byte-order for the machine you are on)
XDOB (double-precision float, in local byte-order for the machine you are on)

You may notice that we sometimes pass around STRINGS TO REPRESENT NUMBERS, like the null-terminated string "123" to represent the number 123.

This is simply to avoid byte-order conversion head-aches.

Now you need to know what the format is to send messages to X-Plane by UDP!

All of the UDP messages have the same format, which is:

5-character MESSAGE PROLOGUE (to indicate the type of message)

and then a

DATA INPUT STRUCTURE (containing the message data that you want to send or receive)

So what is the 5-char message prologue? Easy!

The first 4 chars are the message type,
the 5th char is a byte of value zero, to null-term the label,
and after that comes the data to send!

So, to send a UDP message to X-Plane, just send:

- the 4-letter label
- a byte of value '0'
- the message data you want to send

NOTE ON STRUCTS: All structs are tightly packed, no extra spacing.

If you want diagnostics on what X-Plane is getting or sending, run X-Plane, go to the SETTINGS menu, OPERATIONS AND WARNINGS screen,

turn on the DUMP NET DATA TO LOG.TXT, and then send your messages.

Then exit X-Plane and open the log.txt file to see what size structs X-Plane expected, versus what you sent.

.....
.....
THE MESSAGES YOU CAN SEND
.....
.....

So now you see how to send messages to X-Plane... your final question is: What are the messages I can send?

Well, following are various DATA STRUCTURES that you can send (right after the 5-char MESSAGE PROLOGUE, each being 4 chars (plus a zero byte) as mentioned above):

.....
.....
LOAD AN AIRCRAFT: ACFN
.....
.....

`const xint net_strDIM=150; // must be short enough to send over the net, long enough to hold a full ACF path from the x-system folder`

```
struct acfn_struct
{
    xint m_acfn_p;
    xchr m_acfn_path_rel[net_strDIM];
    xchr pad[2];
    xint m_acfn_live_ind;
};
```

Use this to load an airplane.
Just enter which plane to load as 'p', and the path of the plane as the path. Send this in and X-Plane will load this plane. Send in 1->19 to load the other planes!

.....
.....

INIT THE AIRPLANE AT SOME LOCATION: PREL

```
.....  
.....  
const xint idDIM=8;    // used for nav aids and airports both, so not in the navt or airp  
classes
```

```
struct PREL_struct  
{  
    init_flt_enum    type_start;  
    xint             p_idx;  
    xchr             apt_id[idDIM];  
    xint             apt_rwy_idx;  
    xint             apt_rwy_dir;  
    xdob             dob_lat_deg;  
    xdob             dob_lon_deg;  
    xdob             dob_ele_mtr;  
    xdob             dob_psi_tru;  
    xdob             dob_spd_msc;  
};
```

Use this to set the airplane at some new location.

The TYPE START enums are:

```
loc_repeat_last      5    // used by ATC and reset-flt  
command  
loc_specify_lle      6    // used by maps and the like  
loc_general_area     7    // used to auto-load new airplanes  
when the number of airplanes is increased  
loc_nearest_apt      8    // used when we are just loading a  
new plane, not wanting to change the location in any way,  
but still starting in a proper manner for the new airplane  
loc_snap_load        9    // for when we are getting  
snapshots from movies and the like, so do not want to reset  
the flt-path or move the plane!  
  
loc_ram              10   // Ramp start  
loc_tak              11   // Takeoff on runway  
loc_vfr              12   // VFR app to runway  
loc_ifr              13   // IFR app to runway  
  
loc_grs              14   // Grass strip
```



```

loc_drt          15 // Dirty strip
loc_grv          16 // Gravel strip
loc_wat          17 // Seaplane start
loc_pad          18 // Helipad start
loc_cat          19 // Carrier cat shot

loc_tow          20 // Glider - Tow Plane
loc_win          21 // Glider - Winch
loc_frm          22 // Formation flying
loc_Are          23 // Refuel - Boom
loc_Nre          24 // Refuel - Basket
loc_drp          25 // B52 drop
loc_pig          26 // Piggy back with shuttle on top

loc_car          27 // Aircraft carrier approach
loc_fri          28 // Frigate approach
loc_rig          29 // Small oil rig approach
loc_pla          30 // Large oil platform approach
loc_fir          31 // Forest fire approach

loc_S01          32 // Shuttle stuff
loc_S02          33 // ""
loc_S03          34 // ""
loc_S04          35 // ""
loc_shuttle_glide 36 // we don't access this one except
by dropping the shuttle when it is in piggyback

```

I would use the “loc_specify_llc” start type to simply set the lat, lon, and elevation as you like.

You can leave the airport info set to ZERO in that case.

```

.....
.....
OK FINE LOAD AN AIRCRAFT>>>AND<<< INIT THE AIRPLANE AT
SOME LOCATION! ACPR
.....
.....

```

Send in “ACPR” plus a NULL plus the TWO structs mentioned above!

```

.....

```

.....
OK NOW LOAD or SAVE A SITUATION OR MOVIE!
.....
.....

Send in "SIMO" plus a NULL plus the struct mentioned below!

```
enum
{
    SIMO_save_sit=0,
    SIMO_load_sit=1,
    SIMO_save_mov=2,
    SIMO_load_mov=3
};

struct SIMO_struct // situation or
movie to load or save... kind of nice and requested
{
public:
    SIMO_struct() {memset(this, 0, sizeof(*this));}
    ~SIMO_struct() {}

    xint SIMO_type; // any of the SIMO_xxx enums listed above
    xchr SIMO_path[net_strDIM];
    xchr pad[2];
};
```

.....
.....
RUN A COMMAND: CMND
.....
.....

DATA INPUT STRUCTURE is a string

The data part of this message is simply the command that you want X-Plane to initiate!

Easy!

To see the (plethora) of X-Plane commands, run X-Plane and go to the Settings menu, Joystick and Equipment screen, Buttons:Advanced tab. The commands are the group name you see in the center of the screen,

PLUS command string in the right side, all run together.
So, for example, the first command in the sim is: "none/none".
OK that was a boring one.
But there are more! Lots more!

TYPICAL MESSAGE TO GET FLAPS ONE NOTCH UP WILL LOOK LIKE

CMND0+sim/flight_controls/flaps_up

```
.....  
.....  
SEND ME ALL THE DATAREFS I WANT: RREF  
.....  
.....
```

So this one is cool: Send in the 5 chars RREF (null-terminated!) plus this struct:

```
struct dref_struct_in  
{  
    xint dref_freq;  
    xint dref_sender_index; // the index the customer is using to define this dataref  
    xchr dref_string[400];  
};
```

Where dref_freq IS ACTUALLY THE NUMBER OF TIMES PER SECOND YOU WANT X-PLANE TO SEND THIS DATA!

Where dref_en is the integer code you want X-Plane to send back with the dataref value so you can tell WHICH dataref X-Plane is giving you! (since you are likely to ask for MANY different datarefs!)

Where dref_string is the dataref string that you want X-Plane to send to you!

And, if the dataref is to an ARRAY of values (like engine thrust, since there can be 8 engines), just add [xxx] to the end, where "xxx" is the array index you want X-Plane to send!

The [and] should simply surround the number to indicate that the number is the index you want.

So, send in "sim/flightmodel/engine/POINT_thrust[1]" to have X-Plane send the second engine, for example (since we start at 0!)

NOTE: You can send at most about 148 requests at one go (that is, without X-Plane advancing one frame in between). If you are requesting more than about 140 data refs, it is best to request them in batches of 100, and wait 100 milliseconds before requesting the next batch.

X-Plane will send the message right back to the IP address and port number you sent the RREF command from!

You will get:

```
struct dref_struct_out
{
    xint dref_sender_index ;
    xflt drefflt_value      ;
};
```

Where `dref_en` is the integer code you sent in for this dataref in the struct above.

Where `drefflt` is the dataref value, in machine-native floating-point value, even for ints!

X-Plane will send as many of these structs right behind one another as there are dataref requests pending at any given time for one receiver IP. This means, if more than one dataref request times out the same time, you get `RREF+0+(dref_struct_out)+(dref_struct_out)+(dref_struct_out)+....` as one UDP packet.

NOTE: X-Plane will send at most 183 `dref_struct_outs` at one go, because that fits into a 1500 byte MTU. After 183 structs send, X-Plane will send a new packet, starting again with the RREF header.

So, of course, you can send in all the RREF messages you want, to get all the dataref values back that you want!

Easy!

Send in a “`dref_freq`” of 0 to stop having X-Plane send the dataref values.

```
.....
.....
SET A DATAREF TO A VALUE: DREF
.....
.....
```

```
const xint strDIM=500; // must be long enough for ATIS! we must not use in Pascal str
```

above 255!

```
struct dref_struct  
{  
    xflt var;  
    xchr dref_path[STRDIM];  
};
```

Use this to set ANY data-ref by UDP! With this power, you can send in any floating-point value to any data-ref in the entire sim!
Just look up the datarefs at <http://www.xsquawkbox.net/>.
Easy!

IMPORTANT: NULL TERMINATION MEANS THE NULL CHARACTER MUST BE PLACED AT THE END OF dref_path THEN SHOULD BE FILLED WITH BLANK
SO YOUR TYPICAL MESSAGE SHOULD LOOK LIKE THIS

DREF0+(4byte byte value)+dref_path+0+spaces to complete the whole message to 509 bytes

AN EXAMPLE TO TURN ON AN ANTI-ICE SWITCH WOULD BE

DREF0+(4byte byte value of 1)+
sim/cockpit/switches/anti_ice_surf_heat_left+0+spaces to complete to 509 bytes

DO NOT ADD ANY + SIGNS. THIS IS JUST TO SHOW THE PARTS OF THE MESSAGE TO BE ADDED AS ONE SINGLE BLOCK

REMEMBER: You can go to the SETTINGS menu, OPERATIONS AND WARNING window, to turn on a diagnostics option that will output what data X-Plane thinks it is getting from you to a log file!
Turn this on during development to see what X-Plane THINKS it is getting from you!

.....
.....
SET A DATA OUTPUT TO A VALUE: DATA
.....
.....

Remember how you can output data from the Data Output Screen?
You can also SET data as well with the DATA message!
You just send in the variables that you want to SET!
(Now, you can NOT set ALL the variables! Mach number, for example, is determined by the speed of the plane... so you cannot change that, for example).

But, to enter radios or control deflections by UDP, simply send the DATA message described below TO X-Plane by UDP, and X-Plane will use those messages for input and control of the sim! You may send joystick deflections to fly the plane with your own hardware, or send in any number of other variables selectable in the data output screen... whatever can come out, you can send right back in with an identical message but the values of the number changed! Easy! (Just realize that some messages will be over-ridden by X-Plane!)

```
const xint VALUES_PER_DATA_ITEM=8;

struct data_struct
{
    dout_line_index_t    index           ;    // which index
to send into the list of choices in the data output screen
    xflt                 data[VALUES_PER_DATA_ITEM] ;    // the 8
values that can be sent for each index
};
```

SEND A -999 FOR ANY VARIABLE IN THE SELECTION LIST THAT YOU JUST WANT TO LEAVE ALONE, OR RETURN TO DEFAULT CONTROL IN THE SIMULATOR RATHER THAN UDP OVER-RIDE.

So, to send in a DATA message to control some value in X-Plane, send in:
"DATA" (4 chars)
0 (1 char of val 0)
data_struct (a filled-in data struct as per above, with struct alignment 4)

Do this, and you should be able to control some of the variables in X-Plane by UDP!

.....
.....

SELECT OR DE-SELECT DATA OUTPUT TO NOT OR COCKPIT
DISLPAY: DSEL/USEL/DCOC/UCOC

.....
.....

DATA INPUT STRUCTURE is any series of XINTs

Now, say that you are writing an add-on or something for X-Plane and you want your motion-platform or cockpit to send in a request to X-Plane to send a bunch of data out like this, because you are getting tired of going into the data output screen and making selections of data to output all the time. In that case you will SEND a packet just like the one above to X-Plane, but the label will be "DSEL". The data will be a series of integers indicating which data output you want! (1 for the first in the list, 2 for the second, etc).

So "DSEL0456" would request that X-Plane send the fourth, fifth, and sixth items in the data output screen many times per second to the IP address listed in the Internet Settings screen. DSEL is in characters, but 4 5 6 are YOUR MACHINE-BYTE-ORDER integers.

Use DSEL to select data to send via UDP output.

Use USEL to UN-select data to send via UDP output.

Use DCOC to select data to the COCKPIT DISPLAY rather than UDP output.

Use UCOC to UN-select data to the COCKPIT DISPLAY rather than UDP output.

.....
.....

SET UP THE INTERNET OPTIONS: ISE4

.....
.....

This allows you to set up internet options for X-Plane without touching it. This is useful if you have a simulator with many displays, and do not want to manually set the IP options for each copy of X-Plane. Simply send this structure:

```
struct ISE4_struct // SET AN IP OUTPUT  
v4
```

```

{
    xint index;
    xchr snd_ip_str[16];           // IP's we are sending to, in English
    xchr snd_pt_str[8];           // ports are easier to work with in STRINGS, and
make this 8 bytes not 6 to avoid byte-spacing issues in th struct.. 8 bytes really does fill
the space
    xint snd_use_ip;              // use various IP's
};

```

And following are a list of the enums for X-Plane 11.00:

```

    if(input<=18)sel=ip_mplayer_00    +input    ;    //
multiplayer!
else if(input<=38)sel=ip_exvis_00    +input-19 ;    //
external visuals!
else if(input==39)sel=ip_master_is_exvis ;    //
master machine, this is an external visual!
else if(input==42)sel=ip_master_is_IOS ;    //
master machine, this is an IOS
else if(input==62)sel=ip_IOS_is_master ;    //
IOS, this is master machine!
else if(input==64)sel=ip_DOUT_ui_set ;    //
data output target
else if(input==71)sel=ip_Xavi_1      ;    //
Xavion 1
else if(input==72)sel=ip_Xavi_2      ;    //
Xavion 2
else if(input==73)sel=ip_Xavi_3      ;    //
Xavion 3
else if(input==74)sel=ip_Xavi_4      ;    //
Xavion 4
else if(input==75)sel=ip_fore_ip_addy ;    //
Foreflight, one IP addy
else if(input==76)sel=ip_fore_broadcast ;    //
Foreflight, broadcast
else if(input==77)sel=ip_control_pad ;    //
X-Plane Control pad for IOS

```

.....
.....

SET UP THE INTERNET OPTIONS: ISE6

.....
.....

This allows you to set up internet options for X-Plane without touching it. This is useful if you have a simulator with many displays, and do not want to manually set the IP options for each copy of X-Plane. Simply send this structure:

```
struct ISE6_struct // SET AN IP OUTPUT v6
{
    xint index;
    xchr snd_ip_str[65]; // IP's we are sending to, in English
    sized to match Windows
    xchr snd_pt_str[6]; // ports are easier to work with in
    STRINGS!
    xchr pad1[1];
    xint snd_use_ip; // use various IP's
};
```

Same index enums as the ASE4 message above!

.....
.....

PLAY A SOUND: SOUN

.....
.....

const xint strDIM=500; // must be long enough for ATIS! we must not use in Pascal str above 255!

```
struct soun_struct // play any sound
{
    xflt freq, vol;
    xchr soun_path[strDIM];
};
```

Use this to simply play a WAV-file sound. Enter the path of the WAV file in the struct. The freq and volume scale 0.0 to 1.0. Easy!

.....
.....

PLAY A LOOPING SOUND: LSND and SSND

```
.....
.....
const xint strDIM=500; // must be long enough for ATIS! we must not use in Pascal str
above 255!
```

```
struct loop_struct
{
    xint index;
    xflt freq, vol;
    xchr soun_path[strDIM];
};
```

Use this to simply play a WAV-file sound THAT LOOPS, with index 0 to 4 (so you can have 5 going at once) LSND starts it, SSND stops it.

```
.....
.....
LOAD AN OBJECT: OBJN
.....
.....
```

```
const xint strDIM=500; // must be long enough for ATIS! we must not use in Pascal str
above 255!
```

```
struct objN_struct // object name: draw any object in the
world in the sim
{
    xint index;
    xchr path_rel[strDIM];
};
```

Just like the airplane struct, but with any OBJ7 object (see the San Bernardino "KSBD_example.obj" sample object in the Custom Scenery folder for an example of an OBJ7 object.

With this message, simply send in the path of any object that you have on the drive and you want X-Plane to display! The location is controlled with the struct below.

```
.....
.....
PLACE AN OBJECT: OBJL
```

```

.....
.....
struct objL_struct // object location: draw any object in
the world in the sim
{
    xint index;
    xchr pad1[4];
    xdob lat_lon_ele[3];
    xflt psi_the_phi[3];
    xint on_ground;           // is this object on the ground? if so, simply enter 0
for the elevation, x-plane will put it on the ground
    xflt smoke_size;        // is this object smoking? if so, simply indicate the size of
the smoke puffs here
    xchr pad2[4];
};

```

Enter the location of the object you loaded here. It can be a tank driving around on the ground, a missile firing, or anything else you can imagine.

```

.....
.....
MAKE AN ALERT MESSAGE IN X-PLANE: ALRT
.....
.....

```

```

struct ALRT_struct           // MAKE AN ALERT
MESSAGE, used between copies of X-Plane
{
public:
    ALRT_struct() {memset(this,0,sizeof(*this));}
    ~ALRT_struct() {}

    xchr m_m1[240]; // needs to be multiple of 8 for the align to work out perfect for
the copy?
    xchr m_m2[240]; // needs to be long enough to hold the strings!
    xchr m_m3[240];
    xchr m_m4[240];
};

```

```

.....
.....

```

FAIL A SYSTEM: FAIL

.....
.....

Fail a system, where the data will indicate which system to fail. The system to fail is sent as an ASCII STRING (ie: "145"), where the 0 is the first failure listed in the failure window in X-Plane (currently the vacuum system) and incremented by 1 from there.

.....
.....

RECOVER A SYSTEM: RECO

.....
.....

Recover a system, where the data will be an integer indicating which system to recover. The system to recover is sent as an ASCII STRING (ie: "145"), where the 0 is the first failure listed in the failure window in X-Plane (currently the vacuum system) and incremented by 1 from there.

.....
.....

FAIL A NAVAID: NFAL

.....
.....

Fail a NAVAID, where the data will be the ID of which NAVAID to fail.

.....
.....

RECOVER A NAVAID: NREC

.....
.....

Recover a NAVAID, where the data will be the ID of which NAVAID to Recover.

.....
.....

RECOVER ALL FAILED SYSTEMS: RESE

.....

.....

Just send RESE followed by a NULL of course to recover ALL failed system.

.....

.....

SHUT IT ALL DOWN. GO HOME. WE'RE DONE.

.....

.....

QUIT (no message needed after this label. we're done here)

SHUT (no message needed after this label. we're done here)

.....

.....

DISCOVER X-PLANE BY A BEACON

.....

.....

In order to send and receive UDP messages to talk to X-Plane, you must know the IP-address of that machine within your network. You can either enter the IP-address manually, or you can use the BEACON that each instance of X-Plane running in your network broadcasts to announce its presence.

The BEACON message uses a mechanism called multicast, which is a special IP-address range that you can subscribe to in order to get announcements from X-Plane. This works both when X-Plane is running on the same local machine, or on a machine in the same local area network.

In order to subscribe to X-Plane's BEACON, you must join the multicast group 239.255.1.1 and listen on port 49707. While this looks like an IP-address that is not in your network, it is really a group identifier. Consult your operating system's documentation on how to join a multicast group with a UDP socket. You will want to read the documentation of the `setsockopt()` function and the `IP_ADD_MEMBERSHIP` parameter.

When you configure a socket to receive X-Plane's multicast messages, you also want to use the `SO_REUSEADDR` (`SO_REUSEPORT` on Mac) option. This is important so that multiple applications on the same machine can all receive the BEACON from X-Plane. If you don't use

SO_REUSEADDR (SO_REUSEPORT on Mac) only one application per machine will be able to detect X-Plane, and others will get a failure when they try to bind the socket.

However, be careful that you normally do NOT want to use any of the REUSE* options for the sockets you receive unicast traffic from X-Plane on. Unless you know exactly what you are doing, DO ONLY set SO_REUSEPORT or SO_REUSEADDR for receiving X-Plane's BEACON on port 49707.

Once you are receiving the BEACON messages from X-Plane, the struct must be interpreted as follows:

5-character MESSAGE PROLOGUE which indicates the type of the following struct as
BECN\0

```
struct becn_struct
{
    uchar beacon_major_version;    // 1 at the time of
X-Plane 10.40
    uchar beacon_minor_version;    // 1 at the time of
X-Plane 10.40
    xint application_host_id;      // 1 for X-Plane, 2 for
PlaneMaker
    xint version_number;          // 104103 for X-Plane
10.41r3
    uint role;                    // 1 for master, 2 for
extern visual, 3 for IOS
    ushort port;                 // port number X-Plane
is listening on, 49000 by default
    xchr    computer_name computer_name[500];    // the
hostname of the computer, e.g. "Joe's Macbook"
};
```

Parsing this struct allows you to find any instance of X-Plane running in the network, find out which version of X-Plane is running, see the name of the computer and find out whether it's configured as a master or visual slave machine, and lastly find out if X-Plane's receive port has been changed from the default of 49000.

You can expect the struct to be compatible within the same major version

of the BEACON. Expect structs to change when the major version changes, so you will want to abort parsing when you discover a mismatch of the beacon_major_version.

.....
.....
DATAREF READ/WRITE
.....
.....

If you go to the SETTINGS menu in X-Plane as selected on the upper-right corner of the screen,
Then to the DATA OUTPUT tab,
Then to the DATAREF READ/WRITE group,
There, you can select any dataref for outputting from X-Plane, or reading into X-Plane, to or from either a networked computer by UPD, or a com-port, as you like.

Here is the format:

Send data to NETWORKED COMPUTER (via UDP):

This struct will be filled in and sent for each dataref selected: (enter the IP address in the GENERAL DATA OUTPUT group right there on that screen)

`const xint strDIM=500; // must be long enough for ATIS! we must not use in Pascal str above 255!`

```
struct DREF_struct
{
    xflt var;
    xchr dref_path[strDIM];
};
```

Send data to A COM PORT:

A string will be built and sent to the com port, which is simply \$xxx.xxx.xxx, etc, where each xxx is the floating-point value of that dataref in the list, in the order shown in the selection list.

For reading, it's the same thing: Have the variables selected so the values read go into the right slots in X-Plane.

OK so there you have it!