

原作者： X-Plane Team

翻译： PeterShen （Skysim Team）

译者团队网站： [www.skysim.cc](http://www.skysim.cc)

版权所有： Skysim Team

日期 : 2017.5.7

翻译版本 : 1.0

••••••••••••••••••••••••••••••••••••••••••••••••••••••••

如果你已经在版本10中使用过并想了解版本11中的改动，请看此本快速迁移手册。

•••••••••••••••••••••••••••••••••••••••••••••••••••••••

X-Plane 11的新消息：

用于加载飞行器的ACFN命令，定位飞行器的PREL命令，加载定位飞行器的ACPR命令已配置了新的结构体，具体见文档说明。

X-Plane 11停用的消息：

PAPT place by airport. 改用 PREL.

PMAP place by airport. 改用 PREL.

VEH1 移动单架飞机. 改用 VEHX.

VEHA 移动所有飞机. 改用 VEHX.

MENU 选择一个菜单. 改用 CMND.

CHAR 选择一个菜单. 改用 CMND.

MOUS 选择一个菜单. 改用 CMND.

ISET 打开或关闭一个英特网输出. 改用 ISE4.

ACFN 旧的ACF结构体没有livery. 改用 ACFN，livery可选项见文档.

HACF 炸掉一个飞行器 . 不再支持.

R\_QT 播放一个quicktime电影. 不再支持.

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

X-Plane的驱动与监听

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

此文档将讲述如何获取与发送数据到X-Plane中。你允许任意处理使用到的数据，包括模拟舱，动平台等。需要使用作者规范的消息去驱动、监听X-Plane，但此行为不受作者约束。

此文档展示了如何通过UDP接口读取、发送信息给X-Plane，包括以太网接口与WIFI。另外，你可以Google下X-Plane的插件，学习如何编写一个运行在X-Plane中的插件。插件可以更快的控制更多参数，不过写插件需要一定的学习成本。作者推荐使用插件去读写大量的数据，这样可以得到更快的响应速度。你也可以不使用插件，使用UDP简单的读写一些数据，以建立你自己的程序。

本文需要你对UDP消息有一定的了解，UDP消息描述才是本文的重点。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

提示

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

提示：X-Plane总是从49000接口接收数据。

提示：所有字符串需要以空值（NULL）结尾。

提示：打开 Setting菜单，找到 OPERATIONS AND WARNINGS 标签， 打开 dump net data to log.txt 按钮，使X-Plane丢弃error.out文件的log数据。打开输出会拖慢一些模拟器速度，不过作为调试用可以打开一会以便观察数据。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

开始使用

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

使用X-Plane数据交互有两个用途：

1. 将X-Plane作为自己仿真模型的视景使用。

2、将X-Plane仿真的数据输入到自己的系统当中。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

输出数据驱动外部程序: RPOS

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

一个从X-Plane获取到的输出数据可以驱动你的自己的程序或地图。

发送以下数据到X-Plane的IP地址，UDP输入端口号为49000：

RPOS（四字符加上一个NULL在尾部，共五字符）加上一个字符串（同样需要以NULL结尾）。字符串包含一个数字，说明你希望X-Plane每秒给你回馈多少个位置信息。

发送 RPOS\_60\_，\_代表了NULL，说明X-Plane每秒将发送60次的位置信息。

注意：X—Plane会将你发送的信息立刻回馈到你发送RPOS指令的地址与端口！

在 Setting 菜单，internet settings标签，最右侧的标签卡中有数据输出。那里显示了应该发送给你的数据，数据被格式化过没有任何多余的空格。

四字符 RPOS 与一个 NULL.

double dat\_lon 经度，度

double dat\_lat 纬度，度

double dat\_ele 海拔高度，米

float y\_agl\_mtr 对地高度，米

float veh\_the\_loc pitch，度

float veh\_psi\_loc 真实航向，度

float veh\_phi\_loc roll，度

float vx\_wrl X轴速度，向东，米/秒

float vy\_wrl Y轴速度，向上，米/秒

float vz\_wrl Z轴速度，向南，米/秒

float Prad roll率，弧度/秒

float Qrad pitch率，弧度/秒

float Rrad yaw率，弧度/秒

这是X-Plane最通常的数据输出。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

获取天气雷达信息: RADR

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

一个输出数据可以让你建立自己的气象雷达。

发送以下数据到X-Plane的IP地址，UDP输入端口号为49000：

五个字符RADR（四个字符加上尾部NULL）加上一个字符串（同样以NULL结尾），声明了X-Plane每秒向你发送多少个雷达数据点。

发送 RADR\_10\_，\_代表NULL，说明X-Plane每秒向你发送10个数据点。

注意：X-Plane会立即将你发送的RADR命令返回给发送数据的IP端口！

同样在标签卡可以看到发送的数据。

四个字符 RADR 和一个 NULL.

float lon 雷达点经度

float lat 雷达点纬度

float storm\_level\_0\_100 降水量，0到100

float storm\_height\_meters 风暴平面的海拔高度，米

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

利用现有数据驱动视景: VEHX

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

利用X-Plane提供更好的视景画面，让现有的数据更好的显示。

发送以下数据到X-Plane的IP地址，UDP输入端口号为49000：

五个字符 VEHX（四个字符加一个NULL）加上数据, 数据格式已格式化过，数据间没有空格。

int p 飞机序号，0为主飞机

double dat\_lat 经度，度

double dat\_lon 纬度，度

double dat\_ele 海拔高度，米

float veh\_psi\_true 航向，度

float veh\_the pitch，度

float veh\_phi roll，度

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

其他消息

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

下面是X-Plane所定义的数据类型：

XCHR 字符

XINT 4 byte int

XFLT 4 byte int 和 float

XDOB 双精度float

（以上数据类型转化根据机器平台所定）

有时候直接使用字符串去替代数字直接传递，避免了byte转化的问题。

所有的UDP包都采用相同的格式：

5字符的开头（定义消息的类型），然后是数据的结构体（包含了你想要发送或者接收的数据）。开头的五个字符中，前四个是消息类型，第五个是一个空值，后面跟上需要发送的数据。

所以，想要给X-Plane发送UDP消息，就发送：

-四个字符的标签

-一个byte的空值

-你想要发送的消息的数据

注意结构体：

你可能通过UDP发送或接收多个结构体，所以记得在代码中声明。

包括在Mac上使用时，结构体字节必须与Intel 64-bit X-Code对齐。对齐方式是：四字节为int或float，八字节为double。所以，一个int也可能占用八个字节，如果结构体中存在一个八字节int，那下一个变量是一个double！

切记，如果结构体中含有double类型的参数，那么一个4byte int的数组中可能包含8byte的double。每个变量还是占用4byte，交互时还是以4byte为分界，只有最初的参数是以8byte来处理的。

不同开发者使用不同编辑器与编辑选项，经常使得数据格式填充不完整，最后导致数据长度错误。

作者建议刚开始调试时，打开设置中的调试窗口查看交互的数据有无问题。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

允许发送的数据

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

以下的数据结构体是允许被发送的（同一为5字符的描述开头）。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

加载一架飞机：ACFN

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

struct acfn\_struct

{

xint acfn\_p ;

xchr acfn\_path\_rel[150] ;

xint livery\_index ;

};

使用这个结构体去加载飞机。

acfn\_p是加载的飞机，path是飞机的路径，发送以上数据可以加载预定的飞机。发送1到19可以加载其他飞机。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

指定地点初始化飞机：PREL

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

struct PREL\_struct

{

init\_flt\_enum type\_start ;// 见下表说明

xint p\_idx ;// 飞机序号，0为自己，1-19为其他飞机

xchr apt\_id[idDIM];//机场ID，需要查询机场的ID

xint apt\_rwy\_idx ;// 跑道序号，需要查询ID

xint apt\_rwy\_dir ;// 跑道方向，需要跑道序号支持

xdob dob\_lat\_deg ;// 经纬高度指定Start，经度

xdob dob\_lon\_deg ;// 纬度

xdob dob\_ele\_mtr ;// 高度

xdob dob\_psi\_tru ;// 航向，度

xdob dob\_spd\_msc ;// 速度，米/秒

};

使用此参数可以让飞机置位在某个地点。

参数Start的值如下：

loc\_repeat\_last 5 // 给ATC或重置飞行使用

loc\_specify\_lle 6 // 给地图使用

loc\_general\_area 7 // 自动加载飞机并增加飞机数量

loc\_nearest\_apt 8 // 加载新飞机，不改变位置

loc\_snap\_load 9 // 从快照中加载，不改变航路与飞机位置

loc\_ram 10 // 斜坡开始

loc\_tak 11 // 跑道上起飞

loc\_vfr 12 // VFR进近

loc\_ifr 13 // IFR进近

loc\_grs 14 // 草地跑道

loc\_drt 15 // 脏跑道

loc\_grv 16 // 砂石跑道

loc\_wat 17 // 水上机位

loc\_pad 18 // 直升机停机坪

loc\_cat 19 // 舰载平台

loc\_tow 20 // 滑翔机，牵引机

loc\_win 21 // 滑翔机，卷扬机

loc\_frm 22 // 编队飞行

loc\_Are 23 // 重新加油- Boom

loc\_Nre 24 // 重新加油 - Basket

loc\_drp 25 // B52投掷

loc\_pig 26 // 坨着航天飞机

loc\_car 27 // 运输机进近

loc\_fri 28 // 护卫舰进近

loc\_rig 29 // 少油进近

loc\_pla 30 // 重油进近

loc\_fir 31 // 森林火灾进近

loc\_SO1 32 // 航天飞机

loc\_SO2 33 // ""

loc\_SO3 34 // ""

loc\_SO4 35 // ""

loc\_shuttle\_glide 36 // 只能用于抛投坨着的航天飞机

通常使用 loc\_specify\_lle 去简单指定经度，纬度，高度加载新飞机。

此情况下你可以将机场参数设置为0。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

加载飞机并指定位置： ACPR

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

发送ACPR加上一个NULL，跟上两个上述的结构体即可。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

运行一个指令: CMND

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

数据输入结构体是一个字符串

初始化消息的数据非常简单！

打开设置，可查看X-Plane的指令，指令被组别化命名了。例如，指令格式为none/none。

典型的获取襟翼抬升的消息如下

CMND0+sim/flight\_controls/flaps\_up

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

发回一个我希望获取到的值: RREF

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

发送5字符的指令 RREF (NULL结尾) 加上一个结构体:

struct dref\_struct\_in

{

xint dref\_freq ;

xint dref\_en ;

xchr dref\_string[400] ;

};

dref\_freq说明每秒发送几次数据。

dref\_en是一个integer参数，为了说明反馈的是哪个dataref，通常你会请求很多个参数。

dref\_string说明你需要什么dataref。

如果dataref是一个数组（比如引擎推力，可能有8个引擎），就在后面加一个[xxx]，xxx是序号。[与]中应简单的包含一个数字去指定你需要的序号。

所以，发送 sim/flightmodel/engine/POINT\_thrust[1]，可以得到第二号引擎的数据（起始编号为0）。

X-Plane将立刻反馈到你发送指令的IP端口。

你将得到：

struct dref\_struct\_out

{

xint dref\_en ;

xflt dref\_flt;

};

dref\_en说明了dataref在结构体中处在的序号。

dref\_flt是dataref值，有可能是浮点数，也可能是整数。

这样你可以获取任何你想要的参数。

发动一个为零的dref\_freq的参数停止收取dataref反馈。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

给一个Dataref发送数据： DREF

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

struct dref\_struct

{

xflt var;

xchr dref\_path[500];

};

使用这个参数设置任意的data-ref。你可以发送任意的float值给整个模拟器中任意的data-ref。可以前往http://www.xsquawkbox.net/查看有关dataref的东西。

注意：一个NULL字符必须放置在dref\_path的结尾，否则会失败，你发送的消息应该是以下格式：

DREF0+（4byte的var值）+dref\_path+0+多余的NULL填满整个509byte的消息

（0应为NULL）

以下是打开除冰开关的样例：

DREF0+(4byte的值，为1)+ sim/cockpit/switches/anti\_ice\_surf\_heat\_left+0+多余的NULL填满整个509byte的消息

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

切勿添加

记住：你可以打开Setting菜单，打开diganostics选项以便查看输出的数据与X-Plane想获取的数据。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

将数据输出设置为值： DATA

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

在输出数据的同时你还可以设置输出数据的值（不是所有的值都可以改变，比如马赫数就不可以改动）。

在从UDP输入参数或控制指令时，X-Plane会使用它们作为模拟数据操作，你可以使用自己的硬件去发送手柄的控制指令从而操控飞机。当立即回传的值改变时，有可能是X-Plane复写了这个参数。

struct data\_struct

{

int index; // 数据编号

float data[8]; // 最高8byte的数据输出，很多值都不会占用整个8byte

};

发送一个-999意味着你不利用这个值，或将操作恢复默认。

所以想发送一个数据消息去控制一些值，要发送：

DATA（4字节）

0（0为一个字符）

data\_struct（数据结构体部分，4byte对齐）

使用以上方式可以对X-Plane中的一些值进行控制。

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

选择参数显示或不显示在座舱界面：DSEL/USEL/DCOC/UCOC

•••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

数据输入结构体是XINT的一个序列。

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).

例如你在驱动一个运动平台时，将有很多的调用的参数显示在屏幕上，这时

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 ; // the type enum to send to

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!

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[46] ; // IP's we are sending to, in english

xchr snd\_pt\_str[ 6] ; // ports are easier to work with in STRINGS!

xint snd\_use\_ip ; // use various IP's

};

Same index enums as the ASE4 message above!

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

PLAY A SOUND: SOUN

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

struct soun\_struct // play any sound

{

xflt freq,vol ;

xchr path[500] ;

};

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

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

struct loop\_struct

{

xint index ;

xflt freq,vol ;

xchr soun\_path[500] ;

};

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

••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

struct objN\_struct // object name: draw any object in the world in the sim

{

xint index ;

xchr path[500] ;

};

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 ;

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

};

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 ASCI 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 ASCI 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[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.

OK so there you have it!