Copilot X

Android application by Appropel

Instruction Manual v2.10

Appropel

INTRODUCTION

Copilot X is an application that allows you to issue voice commands through your Android device to control X-Plane, the popular flight simulator developed by Laminar Research. The application works by using the Google Voice Recognition service to process your spoken commands and send the appropriate data to X-Plane. Now, complicated procedures such as dialing in a radio frequency or setting the autopilot hold altitude are as simple as saying what you want! Garmin is currently working on voice-recognition systems for real aircraft such as the Lear 70, so this app is a glimpse into the future.

CONFIGURATION

Activate the Settings menu item to configure Copilot X. The first and most important parameter is the IP address of the computer which is running X-Plane. If you are not sure what the address is, it can be found in X-Plane under the Settings->Net Connections menu. Once the proper address has been configured Copilot X is ready to send commands to X-Plane. Note that currently there is no field to configure the port in use - Copilot X will transmit to the default port used by X-Plane, which is 49000.

The second parameter identifies the aircraft that you are flying in X-Plane. The Generic setting will work with most aircraft. If you are flying the x737 by EADT you may select it in order to enable additional commands that are specific to this aircraft (see the full list of commands at the end of this manual). Where there is a conflict the commands for this specific aircraft override the generic ones.

As of version 2.00 you may create and load your own command sets. This opens up the possibility of

- modifying the default command set
- supporting custom aircraft that have additional datarefs
- translating commands into other languages

Refer to Appendix A for further information on custom command sets.

Copilot X has been developed and tested with X-Plane 9 and 10. Some commands may work on earlier versions but this is not guaranteed.

OPERATION

When Copilot X is in the foreground the entire screen of the Android device becomes a button. This makes it easy to activate Copilot X without having to look at the screen. Touch *and hold* the button to issue a command; while

continuing to press the screen, speak your command in a loud and clear voice. Release the button once you are finished speaking. You will then hear either a chime, indicating your command was understood, or a buzz, indicating that it was not.

Copilot X uses Google's Voice Recognition service and it is tied to the performance of that service. If your command is not understood, try speaking more slowly with clear pauses between words. Some terms used in aviation are difficult for Google to recognize, and Copilot X does its best to identify possible mistakes and still determine the phrase. Google will not recognize certain abbreviations, thus these must be spoken as full phrases; see the list of commands for details. It is worth noting that the performance of Google's voice recognition was substantially enhanced with the release of Jellybean (Android 4.1); on such devices (for example the Nexus 7 tablet) Copilot X responds very well.

EXTPLANE PLUGIN (OPTIONAL)

Copilot X supports the ExtPlane plugin by Ville Ranki (https://github.com/vranki/ExtPlane). When this plugin is installed into X-Plane additional features become available in Copilot X:

- New commands such as flaps up/down one notch at a time
- Use of a button on the joystick to activate voice recognition
- GPS/FMS programming

Building and installing plugins is an advanced topic; if you are not comfortable with this we do not recommend trying it! ExtPlane is optional and is not required for most of the functionality in Copilot X. Once the plugin is installed in your X-Plane directory, visit the Settings page in Copilot X and activate the "Use ExtPlane plugin" checkbox. To use a joystick button as voice activation (similar to tapping the Android screen, but keeping hands on yoke!) enter the button number in the "Push-to-talk button number" preference. The button number can be determined in X-Plane using the Joystick and Equipment screen.

If the 'Run in background' option is checked, Copilot X will continue to listen for voice commands when other applications are in the foreground. This is only useful when using a mapped button with ExtPlane as described above (but it is *very* useful, as the pilot can keep both hands on the yoke and eyes on the screen!).

ADDITIONAL FEATURES

Two additional features can be activated in the "Other Options" section of the preferences:

Show recognized text - when this option is checked the app will display the text that was recognized by Google's Voice Recognition service. The phrase that was actually matched as a command will be highlighted in green. This option can be useful for understanding what Google thinks you are saying, for example when developing command sets.

Read back commands - when this option is checked the app will read back all commands, just like a copilot would in real life.

LIST OF COMMANDS

Copilot X understands over 150 distinct spoken commands, allowing you to control almost every knob and switch in the typical X-Plane cockpit. Commands always begin by identifying the item to be manipulated, followed by the state to set the control to. Some commands, such as those that deal with radio or autopilot settings, end with a number. Numbers may be spoken conventionally or spelled out digit by digit; either way should work.

The following is a complete list of available commands, grouped by system.

GENERIC AIRCRAFT

ADF

A-D-F one frequency *number* (e.g. 316)
A-D-F two frequency *number*
A-D-F one heading *number* (e.g. 140)
A-D-F two heading *number*
or
direction finder one frequency *number*
direction finder two frequency *number*
direction finder one heading *number*
direction finder two heading *number*

ALTIMETER

altimeter *number* (e.g. 3011)

APU

A-P-U off/on/start

ARRESTING HOOK

arresting hook extend/retract

AUTOBRAKE

autobrake rejected takeoff/off/1/2/3/max

AUTOFEATHER

auto feather off/on/test

AUTOPILOT

Note that as of v1.20 the word "engage" is no longer used to activate most autopilot functions. Speaking the name of the button acts as a toggle (the only exception to this is the back course button).

autopilot altitude *number* (e.g. 6000)
autopilot altitude flight level *number* (e.g. 360)

autopilot altitude

autopilot speed *number* (e.g. 140)

autopilot autothrottle

autopilot back course
autopilot disengage back course

autopilot flight level change

autopilot glide slope

autopilot heading *number* (e.g. 220)

autopilot heading

autopilot localizer

autopilot vertical speed one thousand
autopilot vertical speed minus five hundred

autopilot vertical speed

BATTERY

battery (1-8) off/on

BRAKES

brakes on/off

CARBURETOR HEAT

carburetor heat on/off

COWL FLAPS

cowl flaps open/closed

DME

D-M-E frequency *number* (e.g. 111.2)
D-M-E mode remote/frequency/groundspeed
D-M-E select 1/2

ECAM

ecam mode engine/fuel/flight controls/hydraulics/failures

FADEC

F-A-D-E-C (1-8) off/on

FLAPS

flaps extend/retract (*ExtPlane required)*
flaps extend/retract full
flaps up/down (*ExtPlane required)*
flaps up/down full

FLIGHT DIRECTOR

flight director off/on/auto

FMS (*ExtPlane required for all commands)*

Note: the FMS can be spelled out as 'f-m-s' or addressed as 'flight'. Identifers are spelled out in standard phonetic code.

flight init
flight previous
flight next
flight clear
flight airport *identifier* (e.g. kilo charlie oscar november)
flight v-o-r *identifier* (e.g. mike hotel tango)
flight n-d-b *identifier* (e.g. charlie oscar)
flight fix/waypoint/intersection *identifier* (e.g. romeo alpha yankee mike yankee)
flight fly at *number* (e.g. five thousand)
flight fly at flight level *number* (e.g. three six zero)

FUEL PUMP

fuel pump (1-8) off/on

FUEL SELECTOR

fuel selector off/left/center/right/both/all

GENERATOR

generator (1-8) off/on

GPS (*ExtPlane required for all commands)*

Note: Identifers are spelled out in standard phonetic code.

g-p-s airport *identifier* (e.g. kilo charlie oscar november)
g-p-s v-o-r *identifier* (e.g. mike hotel tango)
g-p-s n-d-b *identifier* (e.g. charlie oscar)
g-p-s fix/waypoint/intersection *identifier* (e.g. romeo alpha yankee mike yankee)

ICING

ice off/on
ice A-O-A off/on
ice inlet off/on
ice pitot 1 off/on
ice pitot 2 off/on
ice prop off/on
ice window off/on
ice wing off/on

IGNITER

igniter (1-8) off/on

IGNITION

ignition off/right/left/both
ignition (1-8) off/run

INTERIOR LIGHTS

cockpit brightness *percentage* (e.g. 70)
hud brightness *percentage*
instrument brightness *percentage*

LANDING GEAR

landing gear extend/retract
landing gear up/down

MAP

map range 2
(*this is the index of the knob position 1-6, **not** the range in miles*)

PARACHUTE

parachute deploy

PRESSURIZATION

cabin altitude *number* (e.g. 5000)

RADIOS

communication one frequency *number* (e.g. 124.3)
communication two frequency *number*
navigation one frequency *number* (e.g. 114.4)
navigation two frequency *number*
navigation one bearing (or O-B-S) *number* (e.g. 140)
navigation two bearing (or O-B-S) *number*

SPEEDBRAKES

speedbrakes extend/retract

SWITCHES

alternator off/on
avionics master off/on
beacon off/on
inverter off/on
landing light off/on
nav lights off/on
prop sync off/on
strobes off/on
taxi light off/on
yaw damper off/on

TRANSPONDER

transponder off/standby/on/altitude
transponder *number* (e.g. 5152)
squawk *number*

x737 by EADT (http://www.benedikt-stratmann.de/)

AUTOPILOT

autopilot a engage/disengage
autopilot b engage/disengage

autopilot altitude intervention

autopilot altitude *number* (e.g. 14000)

autopilot flight level *number* (e.g. 360)

autopilot altitude

autopilot approach

autopilot arm/disarm autothrottle

autopilot heading
autopilot heading *number* (e.g. 220)

autopilot level change

autopilot lateral navigation

autopilot n-one mode

autopilot speed
autopilot speed *number* (e.g. 210, in knots)
autopilot speed mach *number* (e.g. 0.77)

autopilot vertical speed
autopilot vertical speed *number* (e.g. 1000)

FLAPS

flaps 0/1/2/5/10/15/25/30/40

FLIGHT DIRECTOR

flight director a off/on
flight director b off/on

INTERIOR LIGHTING

c-d-u brightness *percentage*

TROUBLESHOOTING

Various things may prevent Copilot X from working properly and communicating with X-Plane. Here are some things to try if the app does not appear to be working:

- The Android device running Copilot X and the PC running X-Plane should be on the same wireless network. Verify that the IP addresses for both machines are on the same subnet.
- Test if the PC can ping the Android device.
- Verify that the chime is heard after a voice command is spoken. If not, it could indicate that the app is not able to connect to Google's voice recognition service.
- A network analyzer such as Wireshark (http://www.wireshark.org) can help determine if packets are reaching the PC. Copilot X sends UDP packets to X-Plane, so there cannot be error-checking on its side.
- A firewall running on the PC (e.g. Windows Firewall) can prevent your Android device from communicating with X-Plane. Turn the firewall off or set up an exception for the Android device.
- Certain wireless routers are known to be problematic.
- If Copilot X is running offline (i.e. Android 4.1+) support for English voice recognition must be installed. Check under *Settings->Language and input->Voice Search->Download offline speech recognition* and install English if it is not already present.
- To verify that ExtPlane is installed and working, Telnet to port 51000 on your X-Plane machine. See the instructions at the plugin developer's Git repository.

FUTURE DEVELOPMENT

- More custom support for 3rd-party aircraft.
- Extra commands using ExtPlane that cannot be done via UDP.
- Possible support for Bluetooth headsets.

Feedback from users is greatly appreciated.

RELATED APPLICATIONS

Be sure to check out X-Plane to GPS, also from Appropel. This app lets you use any mapping application as a moving map for X-Plane (e.g. Google Maps, Google Earth, Garmin Pilot)!

CREDITS

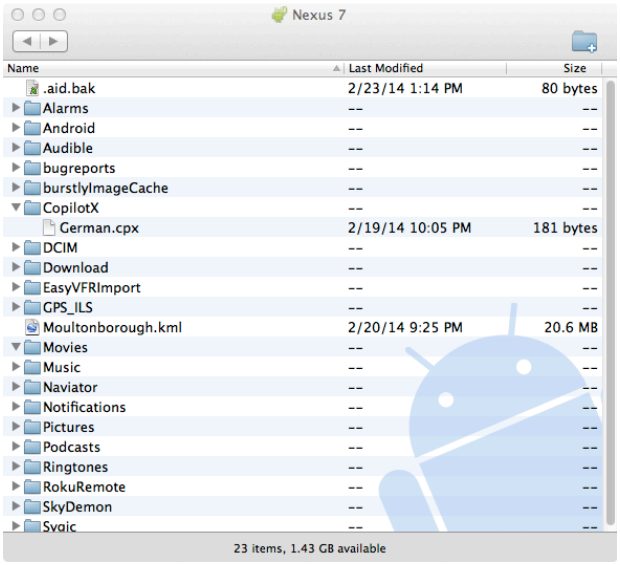
Programming - Kevin Roll / Appropel
Artwork - Heather Lynn / Gossamer Hollow Graphic Design

SUPPORT

Please contact the developer at kroll@appropel.com with any problems, suggestions, or omissions. Appropel is driven by customer feedback and would love to hear from you. Also, reviews on Google Play are greatly appreciated.

APPENDIX A - Writing Custom Command Sets

In order to load a custom command set you will need a program such as Android File Transfer that can access the sdcard storage space on your Android device. This is where you will find the usual directories such as Android, DCIM, Movies, Music, etc. Create a directory named 'CopilotX' at the top level and place your custom command set files there:



Command files in this directory will show up in Copilot X when selecting a Command Set on the preferences screen.

The base command sets that ship with Copilot X can be found in the following Web directory:

http://appropel.com/CopilotX/

These are available for study and to use as the basis for new command sets.

A command set file has the extension '.cpx' and follows a strict format. Here is a portion of the default Generic command set, along with commentary:

```
# Generic command set
```

> *Lines that begin with # are treated as comments, similar to the convention in many programming languages.*

```
name "Generic"
```

```
locale "en-US"
```

These two lines are required in every command set. The name is what will be displayed within Copilot X when selecting a Command Set. The locale is defined by the set of IETF language tags and will be passed to Google's voice recognition service. Thus, when defining a command set in another language, set the appropriate locale here.

```
import "OtherFile.cpx"
```

A command set may contain one or more import statements in order to include other existing command sets. The name in quotes is the actual file name of the command set, and the file must exist either within the application or in the sdcard/CopilotX directory. All definitions within the current file will override existing ones in the imported file, allowing new commands to be layered on top of existing sets. It is OK to have zero import statements if this is a base command set.

```
# Definitions

A = "a"
ADF_ONE = "a d f 1|a t f 1|direction finder 1"
```

The definitions section defines a set of tokens that will be used to parse the strings returned by Google Voice Recognition service. Each value on the right side is a Java regular expression and can contain multiple possibilities separated by the pipe character. Note that Google frequently returns strings that are close but not quite accurate; for better results it pays to include similar-sounding alternatives.

When the voice recognition service returns one or more candidate strings, they are parsed into tokens using the definitions in this section. The tokens are then interpreted as commands using the lines in the next section.

```
# Commands

AUTOPILOT ALTITUDE NUMBER : "handler.setDrefNumber('sim/cockpit/
autopilot/altitude')"
AUTOPILOT ALTITUDE FLIGHT_LEVEL NUMBER :
"handler.setDrefNumberWithFactor('sim/cockpit/autopilot/
altitude', 100)"
AUTOPILOT ALTITUDE : "handler.setDrefValue('sim/cockpit/
autopilot/autopilot_state', 16384)"
```

In the final section the tokens are interpreted to form a complete command. Each definition consists of one or more tokens, a colon, and an action string to be executed. If you are simply changing the command words or translating into another language you will not be interested in what the action strings mean; simply make sure they are preserved verbatim. The programming details of the action strings are given below.

If you are writing a command set to support a custom aircraft you will need to understand how the action strings work in order to address the custom datarefs. Action strings are written in the JEXL scripting language. The JEXL execution context contains a single object named 'handler', which supports the following methods:

`void setDrefValue(final String datarefPath, final float value)`

This is the most basic way to set a dataref value - the parameters are self-explanatory. For example:

```
BRAKES ON : "handler.setDrefValue('sim/flightmodel/controls/parkbrake',
1)"
```

X-Plane's default datarefs are listed in the text file Datarefs.txt in the Resources/plugins folder.

`void setDrefNumber(final String datarefPath)`

*This sets the value of the given dataref. The last token parsed **must** be a NUMBER, and this value will be parsed and provided. The parsing can handle some special cases, e.g. "to"->"2", "niner"->"9" (but currently only in English). For example:*

```
AUTOPILOT HEADING NUMBER : "handler.setDrefNumber('sim/cockpit2/
autopilot/heading_dial_deg_mag_pilot')"
```

`void setDrefNumberWithFactor(final String datarefPath, final float factor)`

Same as setDrefNumber except that the numeric value is multiplied by the given factor. For example:

```
COM_ONE FREQUENCY NUMBER : "handler.setDrefNumberWithFactor('sim/
cockpit/radios/com1_freq_hz', 100)"
```

```
void setDrefNumberWithOffset(final String datarefPath, final
float offset)
```

Same as setDrefNumber except that the numeric value is added to the given offset. For example:

```
AUTOBRAKE NUMBER : "handler.setDrefNumberWithOffset('sim/cockpit/
switches/auto_brake_settings', 1)"
```

```
void setDrefPercentage(final String datarefPath)
```

Calls setDrefNumberWithFactor using a fixed factor of 0.01. In other words the given value is converted from a percentage to a decimal value. For example:

```
COCKPIT BRIGHTNESS NUMBER : "handler.setDrefPercentage('sim/cockpit/
electrical/cockpit_lights')"
```

```
void setDrefNumberWithFactorAndOffset(final String datarefPath,
final float factor, final float offset)
```

A variant of setDrefNumber that allows both a factor and an offset to be applied to the parsed value. There are currently no examples in the existing command sets as this method is only used internally, but it could be useful.

```
void setDrefMappedValue(final String datarefPath, final
Map<Object, Object> map)
```

In this variant a map of possibilities is provided. The parsed number is used as the key into the map and the resultant value is assigned to the dref. For example, here is how the flap settings are mapped on the x737:

```
FLAPS NUMBER : "handler.setDrefMappedValue('x737/systems/flaps/
flapHandleReq', { 0:0.0, 1:0.125, 2:0.25, 5:0.375, 10:0.5, 15:0.625,
25:0.75, 30:0.875, 40:1.0 })"
```

So, the pilot can command any existing setting from 0 to 40, and this is mapped into an actual value from 0-1.

```
void setDrefMultiEngineValue(final String datarefPath, final
float value)
```

In this variant the second token must be a NUMBER, and this value must be in the range 1-8. The index is then decremented by one and appended to the dref path in brackets. For example:

```
FUEL_PUMP NUMBER ON : "handler.setDrefMultiEngineValue('sim/cockpit2/
engine/actuators/fuel_pump_on', 1)"
```

If the user commands "Fuel pump two on", this would assign the value 1 to the dref `sim/cockpit2/engine/actuators/fuel_pump_on[1]`.

`void sendKeyCommand(final String keyCommand)`

For this method to have any effect the user must have the ExtPlane plugin configured and selected in the Copilot X preferences. The given key command will be transmitted to the ExtPlane plugin (by contrast, all of the methods above communicate directly to X-Plane). For example:

```
FLAPS EXTEND : "handler.sendKeyCommand('XPLM_KEY_FLAPSDN')"
```

The complete list of key commands can be found in the XPLMUtilities documentation, however note that Copilot X defines these values in ***upper case*** *following the Java enumeration naming convention.*

`void sendCommand(final String command)`

For this method to have any effect the user must have the ExtPlane plugin configured and selected in the Copilot X preferences. The given command will be transmitted to the ExtPlane plugin. For example:

```
FMS INIT : "handler.sendCommand('sim/FMS/init')"
```

X-Plane's default commands are listed in the text file Commands.txt in the Resources/plugins folder.

`float getNumber()`

Looks for a NUMBER as a token in the command set. If one is found the value is returned, otherwise zero is returned.

`String toFmsString(float number)`

Returns the given number formatted as a 5-digit number for use in the FMS (using the pattern %05.0f).

`String getIdentifier()`

Looks for an IDENTIFIER as a token in the command set. If one is found it is returned, otherwise an empty string is returned.

`void sendToFms(String str)`

Sends the given String to the FMS by translating each letter/digit into a key command.

For reference the complete CommandInterface class is available at the Web directory referenced above.