SASL Reference Manual

Table of Contents

Components Reference

Common Properties

By default all components have a set of default properties described below. Some of them only effect panels but most are common for all components.

size

Type: array of 2 numbers.

Internal size of the component. The first number is the width and the second number is the height. This is a very special property. You can't override it like other properties. It is actually not a property at all but an ordinary variable. You can't use the get function to access it.

position

Type: array of 4 numbers.

The position property contains 4 numbers: {x, y, width, height}. It specifies the position where the component should be drawn.

name

Type: string

Contains the name of this component. Similar to size it is just a string variable, not a property.

visible

Type: boolean

It is false if the component is hidden. For it to be true the component must be visible on the screen. By default all components are visible but all panels are hidden.

movable

Type: boolean

It is true if the component can be dragged by the mouse. By default it is false for all of the components except the panels.

resizeble

Type: boolean

It is true if the component can be resized by the mouse. By default it is false for all components except the panels.

resizeProportional

Type: boolean

It is true if during the resize, SASL will try to keep the component proportions. By default it is true.

focused

Type: boolean

It is true if the component is active (the user clicked on it recently). Focused components receive keyboard input events.

savePosition

Type: boolean

Available in panels only. It is true if the position of this component must be saved in the file when the aircraft is unloaded. Position saving will not work if the name of the panel is not specified.

noBackground

Type: boolean

Available in panels only. If it is true, the panel will be drawn without background. The default value is false.

noClose

Type: boolean

It is available only in panels. If it is true, the resize spot will not be drawn. The default value is false.

noResize

Type: boolean

It is available only in panels. If it is true, resize spot will not be drawn. The default value is false.

Components Callbacks

onMouseClick(component, x, y, button, parentX, parentY)

Called when the mouse button is clicked. The argument “component” contains reference to the component itself. X and Y are the coordinates of the mouse pointer in this component coordinate system. ParentX and parentY are the coordinates of the mouse pointer in the parent component coordinate system. The argument button contains the code of the pressed mouse button (doesn't work in X-Plane properly).

onMouseDown(component, x, y, button, parentX, parentY)

Called when the mouse button is pressed. The argument “component” contains reference to the component itself. X and Y are the coordinates of the mouse pointer in this component coordinate system. ParentX and parentY are the coordinates of the mouse pointer in the parent component coordinate system. The argument button contains the code of the pressed mouse button (doesn't work in X-Plane properly).

onMouseUp(component, x, y, button, parentX, parentY)

Called when the mouse button is depressed. The argument “component” contains reference to the component itself. X and Y are the coordinates of the mouse pointer in this component coordinate system. parentX and parentY are the coordinates of the mouse pointer in the parent component coordinate system. The argument button contains the code of the depressed mouse button (doesn't work in X-Plane properly).

onMouseMove(component, x, y, button, parentX, parentY)

Called when the mouse pointer is moved. The argument “component” contains reference to the component itself. X and Y are the coordinates of the mouse pointer in this component coordinate system. parentX and parentY are the coordinates of the mouse pointer in the parent component coordinate system. The argument button contains the code of the pressed mouse button (doesn't work in X-Plane properly).

onKeyDown(component, char, key)

Called when the keyboard button is pressed. The argument “component” contains reference to the component itself. The argument char contains the pressed character if the button is alphanumeric. The argument key contains the scan code of the pressed button.

This function must return true if the key was processed and shouldn't be passed to other handlers.

onKeyUp(component, char, key)

Called when the keyboard button is depressed. The argument “component” contains reference to the component itself. The argument char contains the depressed character if the button is alphanumeric. The argument key contains the scan code of the depressed button.

This function must return true if the key was processed and shouldn't be passed to other handlers.

draw(component)

Called when the component should draw to itself. The argument “component” contains reference to the drawn component itself. This function may be called several times per frame. It is important to avoid any calculation inside of this function to make your aircraft faster. Use the “update” function for calculations.

Warning: draw callback responsible for calling “draw” of its subcomponents. Use the drawAll(components) function to call draw callback of all subcomponents.

update(component)

Called once for each frame. The argument “component” contains reference to the updated component itself.

Warning: update callback responsible for calling “update” of its subcomponents. Use the updateAll(components) function to call update callback of all subcomponents.

onAvionicsDone()

Called when the aircraft is unloaded. Use this callback to finalize your aircraft (save state, configs, etc).

Default Components

button

Button is a simple component with texture and clickable area. When the mouse is moved over the button it changes its shape to indicate that the button can be clicked.

Properties:

- image – texture drawn on button.

clickable

Clickable area is an empty component with the frame hidden most of the time. When the user select s'show clickable areas' option in the simulator, the clickable area will show a yellow frame.

No additional properties exist in this component.

digitstape

Digits tape shows the numbers similar to an old mechanical instrument. It uses texture with all of the digits. The tape can display both integer and fractional values. There is an option that leading zeros can be added.

Properties:

- image – texture with digits.
- overlayImage – texture will be drawn over the tapes. Absent by default.
- value – number to display
- digits – number of digits to display. Default is 1.
- fractional – number of fractional digits to display. Default is 0.

- allowNonRound – allows the display of fractional numbers. The default is false.
- valueEnabler – if equals to false number will be hidden. The default is false.
- showLeadingZeros – shows leading zeros. The default is false.
- showSign – shows the sign instead of the first digit. The default is false.

frame

Shows the white frame for debugging.

freeTexture

Draws texture inside of this component in the specified position.

Properties:

- image – the texture to draw.
- positionX – the position of the texture inside of the component relative X axis (0..100)
- positionY – the position of the texture inside of the component relative Y axis (0..100)
- width – the width of the texture in the component coords (0..100)
- height – the height of the texture in the component coords (0..100)

lamp

General 2-state indicator. Switches between light depending of state variable.

Properties:

- lightOn – the texture of the lamp is in ON state
- lightOff – the texture of the lamp is in OFF state
- state – the state of the lamp. If it equals true, the lightOn texture will be drawn or lightOff otherwise.

needle

Draws the needle texture rotated by a specified angle.

Properties:

- angle – the angle of the needle rotation in degrees.

- image – needle texture

rectangle

Draws the rectangle of a specified color.

Properties:

- color – rectangle color. It is an array of 4 values: red, green, blue, alpha. All values in range of 0..1. Default value is { 0.15, 0.15, 0.15, 1.0 }

rotary

Rotary knob. It allows the change of value associated with the rotary knob.

Properties:

- image – texture to draw.
- value – value to change.
- step – how much to add to value for each mouse click. Default is 1.
- autoRepeat – allows to change value automatically while mouse button is pressed. Default is true.
- adjuster – callback function. Use it to change value according to your logic, for example to limit value to some range. Callback called with one argument – current value. It should return new value.

rotatedTape

Shows part of texture rotated by specified angle.

Properties:

- image – texture to show.
- window – size of texture part to show. Array of two numbers which are width and height.
- scrollX – horizontal offset of texture.
- scrollY – vertical offset of texture
- angle – rotation angle.

switch

Shows 2-state switch. Switches images depending on state variable. Switch highlighted as clickable.

Properties:

- btnOn – texture to use in ON state
- btnOff – texture to use in OFF state
- state – state of switch. If equals to true btnOn texture will be drawn or btnOff otherwise.

tape

Shows part of texture.

Properties:

- image – texture to show.
- window – size of texture part to show. Array of two numbers which are width and height.
- scrollX – horizontal offset of texture.
- scrollY – vertical offset of texture

textureLit

Draws texture independently of cockpit lighting system. Use this component to simulate lamps in cockpit.

Properties:

- image – texture to draw.

texture

Draws texture.

Properties:

- image – texture to draw.

togglePanelButton

Toggles pop-up panel.

Properties:

- panel – pop-up panel to toggle
- image – button texture

Functions Reference

Properties

defineProperty(name, default)

Define property with specified name and assign default value to it. Use constants as default value or reference to simulator properties. Name must use the same name conventions as all Lua variables.

globalPropertyd(name, default)

Returns reference to simulator property of type double. Default value will be returned by get function in case property is not found.

createGlobalPropertyd(name, default)

Create new simulator property of type double and assign default value to it. Returns reference to created property.

createFuncPropertyd(name, getter, setter)

Create new simulator property of type double with getter and setter callbacks. Returns reference to created property. Getter callback is function without arguments that returns value of type double. Setter function takes one argument of type double.

globalPropertyf(name, default)

Returns reference to simulator property of type float. Default value will be returned by get function in case property is not found.

createGlobalPropertyf(name, default)

Create new simulator property of type float and assign default value to it. Returns reference to created property.

createFuncPropertyf(name, getter, setter)

Create new simulator property of type float with getter and setter callbacks. Returns reference to created property. Getter callback is function without arguments that returns value of type float. Setter function takes one argument of type float.

globalPropertyi(name, default)

Returns reference to simulator property of integer type. Default value will be returned by get function in case property is not found.

createGlobalPropertyi(name, default)

Create new simulator property of integer type and assign default value to it. Returns reference to created property.

createFuncPropertyi(name, getter, setter)

Create new simulator property of integer type with getter and setter callbacks. Returns reference to created property. Getter callback is function without arguments that returns value of integer type. Setter function takes one argument of integer type.

globalPropertys(name, default)

Returns reference to simulator property of string type. Default value will be returned by get function in case property is not found.

createGlobalPropertys(name, maxLength, default)

Create new simulator property of string type and assign default value to it. maxLength is maximum allowed string length for this property. Returns reference to created property.

createFuncPropertys(name, getter, setter, maxLength)

Create new simulator property of string type with getter and setter callbacks. maxLength is maximum allowed string length for this property. Returns reference to created property. Getter

callback is function without arguments that returns value of string type. Setter function takes one argument of string type.

get(property)

Returns value of property.

set(property, value)

Sets property value.

Commands API

findCommand(name)

Find command by name. Returns handler of command or nil if not found.

commandBegin(commandId)

Starts command execution. Obtain commandId by findCommand function.

commandEnd(commandId)

Finish command execution. Obtain commandId by findCommand function.

commandOnce(commandId)

Start and finish command immediately. Obtain commandId by findCommand function.

createCommand(name, description)

Creates new command. Returns command handler if command was successfully created or nil in case of errors.

registerCommandHandler(commandId, before, callback)

Add handler to command. If before equals to 1, handler will be added at the beginning of handlers list. Obtain commandId by findCommand function.

Command handler is function with one argument “phase”: function commandHandler(phase). Phase equals to 0 when command started, 1 when command execution continues and 2 when command is finished. Command handler must return 0 to stop further command processing or 1 to allow more handlers to do its job.

unregisterCommandHandler(commandId, before)

Removes handler added before with registerCommandHandler function.

Components

subpanel(table)

Creates new pop-up component. In table specify values of following properties: name, position, savePosition, noBackground, noClose, noResize, components, command and description.

Command and description are special properties that allow to automatically create command for panel pop-up. Value of command is name of command, value of description will be shown in configuration dialogue.

Components property contains array of panel components.

updateAll(components)

Call update callback for all components in specified array.

drawAll(components)

Call draw callback for all components in specified array.

addSearchPath(path)

Add path to search path array. New path will be added at the very beginning of array.

Graphics

loadImage(fileName)

Loads image into memory. Returns texture handle or nil if texture not found. Image searched according to searchImagePath variable.

loadImage(fileName, width, height)

Loads image part into memory. This function loads central part of texture with specified width and height. Returns texture handle or nil if texture is not found. Image searched according to searchImagePath variable.

loadImage(fileName, x, y, width, height)

Loads image part into memory. This function specified part of texture. Returns texture handle or nil if texture not found. Image searched according to searchImagePath variable.

loadImageFromMemory(data)

Loads image from data string. You can safely free data after calling this function. Returns texture handle or nil if texture wasn't loaded. WARNING: this function doesn't cache images. Use it this caution.

loadImageFromMemory(data, width, height)

Loads image part from data string. You can safely free data after calling this function. This function loads central part of texture with specified width and height. Returns texture handle or nil if texture is not found. WARNING: this function doesn't cache images. Use it this caution.

loadImageFromMemory(data, x, y, width, height)

Loads image part from data string. You can safely free data after calling this function. This function specified part of texture. Returns texture handle or nil if texture not found. WARNING: this function doesn't cache images. Use it this caution.

unloadImage(texId)

Free image from memory. Because of cache image can still remain in memory after calling of this function for some time. Do not use texture handle of unloaded texture anymore or simulator will crash.

getAircraftPaint()

Returns handle of aircraft paint texture.

It is not valid to use this handle in drawing functions until you will call recreateImage function. You can use this texture as rendering target to make some fx.

For better performance do not call this function every frame but find texture once aircraft loaded and save its handle in variable.

getAircraftLiteMap()

Returns handle of aircraft lite map texture.

It is not valid to use this handle in drawing functions until you will call recreateImage function. You can use this texture as rendering target to make some fx.

For better performance do not call this function every frame but find texture once aircraft loaded and save its handle in variable.

findImage(width, height)

Looks for texture in memory with specified width and height. Returns image handle if image is found or nil if not found.

It is not valid to use this handle in drawing functions until you will call recreateImage function. You can use this texture as rendering target to make some fx.

For better performance do not call this function every frame but find texture once aircraft loaded and save its handle in variable.

findImage(width, height, red, green, blue, alpha)

Looks for texture in memory with specified width and height and pixel of specified colour in low left corner. Returns image handle if image is found or nil if not found. Because of image often compressed in memory by quality-loss algorithms, it is better to draw large enough marker square in corner (not just single pixel) to make sure it will not be distorted by compression algorithms.

It is not valid to use this handle in drawing functions until you will call recreateImage function. You can use this texture as rendering target to make some fx.

For better performance do not call this function every frame but find texture once aircraft loaded and save its handle in variable.

recreateImage(texId, width, height)

Destroy old image in memory and create new empty image. Content of image is not defined after this function call. Use it to increase resolution of images you are going to rendering to.

For better performance do not call this function every frame but resize your render target once you've found it.

setRenderTarget(texId)

Start rendering into texture. Always call restoreRenderTarget after this function.

restoreRenderTarget()

Finish rendering to texture and continue rendering to default render target.

getTextureSize(texId)

Returns width and height of texture in pixels.

loadFont(fileName)

Loads font texture. Returns texture handle or nil if texture not found. Font searched according to searchImagePath variable.

drawFrame(x, y, width, height)

Draw one-pixel white frame. Use this function for debugging only.

drawTexture(texId, x, y, width, height)

Draws texture image on screen at position specified by coordinates (x, y). Texture will be scaled to specified width and height. Texture will be mixed with current background colour.

drawTexture(texId, x, y, width, height, red, green, blue, alpha)

Draws texture image on screen at position specified by coordinates (x, y). Texture will be scaled to specified width and height. Texture will be mixed with specified colour components (red, green, blue and alpha).

drawRotatedTexture(texId, angle, x, y, width, height)

Draws texture image on screen at position specified by coordinates (x, y). Texture will be rotated by specified angle in degrees relative to texture centre. Texture will be scaled to specified width and height. Texture will be mixed with current background colour.

drawRotatedTexture(texId, angle, x, y, width, height, red, green, blue, alpha)

Draws texture image on screen at position specified by coordinates (x, y). Texture will be rotated by specified angle in degrees relative to texture centre. Texture will be scaled to specified width and height. Texture will be mixed with specified colour components (red, green, blue and alpha).

drawTexturePart(texId, x, y, width, height, tx, ty, twidth, theight)

Like drawTexture, but only part of the texture will be drawn. Use arguments tx, ty, twidth and theight to specify which part of the texture you'd like to draw. Texture will be mixed with the current background colour.

drawTexturePart(texId, x, y, width, height, tx, ty, twidth, theight, red, green, blue, alpha)

Like drawTexture, but only part of texture will be drawn. Use arguments tx, ty, twidth and theight to specify which part of the texture you'd like to draw. Texture will be mixed with specified colour components (red, green, blue and alpha).

drawRotatedTexturePart(texId, angle, x, y, width, height, tx, ty, twidth, theight)

Like drawRotatedTexture, but only part of the texture will be drawn. Use arguments tx, ty, twidth and theight to specify which part of the texture you'd like to draw. Texture will be mixed with current background colour.

drawRotatedTexturePart(texId, angle, x, y, width, height, tx, ty, twidth, theight, red, green, blue, alpha)

Like drawRotatedTexture, but only part of the texture will be drawn. Use arguments tx, ty, twidth and theight to specify which part of the texture you'd like to draw. Texture will be mixed with the specified colour components (red, green, blue and alpha).

drawRectangle(x, y, width, height, red, green, blue, alpha)

Draws rectangle at the specified position. Rectangle filled by the specified colour components (red, green, blue and alpha).

drawTriangle(x1, y1, x2, y2, x3, y3, red, green, blue, alpha)

Draws triangle filled by the specified colour components (red, green, blue and alpha).

drawLine(x1, y1, x2, y2, red, green, blue, alpha)

Draws 1-pixel wide line of the specified colour.

drawText(font, x, y, string)

Draws text at the specified position. Font texture will be mixed with current background colour.

drawText(font, x, y, string, red, green, blue, alpha)

Draws text at specified position. Font texture will be mixed with specified colour components (red, green, blue and alpha).

Logs

logDebug(str)

Write string str into log with log level 'debug'.

logInfo(str)

Write string str into log with log level 'info'.

logWarning(str)

Write string str into log with log level 'warning'.

logError(str)

Write string str into log with log level 'error'.

Binary Operations

bitand(a, b)

Returns bitwise AND operation of two integer numbers.

bitor(a, b)

Returns bitwise OR operation of two integer numbers.

bitxor(a, b)

Returns bitwise XOR operation of two integer numbers.

Utility

listFiles(path)

Returns array of files and directories located in the specified files. Each array entry is a table with two fields: field 'name' contains name of file or directory. Field 'type' contains string “dir” or “file” depending on the entry type.

toboolean(value)

Convert value to true or false. Like in C language 0 equals to false.

isFileExists(fileName)

Returns true if the file exists.

extractFileName(filePath)

Extract the name of file from the specified full path.

openFile(fileName)

Execute specified lua script. File will be searched like other components according to searchFile variable.

include(fileName)

Execute lua script in context of current component. File will be searched like other components according to searchFile variable. Use this function to make your code structured.

Sound

loadSample(fileName)

Loads sample from file. Returns sample ID you can use in other sound functions. On errors returns 0. Use only mono samples for 3D sounds or 3D sound will not work.

unloadSample(sampleId)

Removes sample from memory. Argument sampleId obtained from the loadSample function.

playSample(sampleId, loop)

Start play of specified sound. Argument sampleId obtained from the loadSample function. If loop argument equals to 0, sound will be played once. If loop argument equals to non-zero, sample will be looped.

stopSample(sampleId)

Stops sound playback. Argument sampleId obtained from the loadSample function.

setSampleGain(sampleId, gain)

Sets gain of sample. Argument sampleId obtained from loadSample function. Argument gain must be in range [0..1000]. 0 is lowest gain, 1000 is maximum gain.

setSamplePitch(sampleId, pitch)

Sets pitch of sample. Argument sampleId obtained from loadSample function. Argument pitch must be in range [0 .. 1000].

rewindSample(sampleId)

Reset sample playback position to zero. Argument sampleId obtained from loadSample function.

isSamplePlaying(sampleId)

Returns non-zero if specified sound playing now. Returns zero if sample is not playing. Argument sampleId obtained from the loadSample function.

setMasterGain(gain)

Adjust gain of all sounds. Argument gain must be in range [0 .. 1000].

setSampleEnv(sampleId, environment)

Setup environment where sound can be heard. Argument sampleId obtained from loadSample function. Argument environment may be one of following constants:

- SOUND_INTERNAL – sound will be heard in cockpit only
- SOUND_EXTERNAL – sound will be heard at external views only
- SOUND_EVERYWHERE – sound will be heard everywhere

getSampleEnv(sampleId)

Returns environment where sound can be heard. Argument sampleId obtained from the loadSample function. Returns one of following constants:

- SOUND_INTERNAL – sound will be heard in cockpit only

- SOUND_EXTERNAL – sound will be heard at external views only
- SOUND_EVERYWHERE – sound will be heard everywhere

setSamplePosition(sampleId, x, y, z)

Sets position of sample relative to the aircraft reference point or to the relative camera (see setSampleRelative function). Argument sampleId obtained from the loadSample function.

getSamplePosition(sampleId)

Returns 3 numbers: x, y, z coordinates of the sample position relative to the aircraft reference point or the relative camera (see getSampleRelative function). Argument sampleId obtained from the loadSample function.

setSampleDirection(sampleId, x, y, z)

Set-up direction of sound. Argument sampleId obtained from the loadSample function.

getSampleDirection(sampleId)

Returns 3 numbers: x, y, z coordinates of sound direction. Argument sampleId obtained from loadSample function.

setSampleMaxDistance(sampleId, maxDistance)

Set maximum distance where sound can be heard. Argument sampleId obtained from loadSample function.

setSampleRolloff(sampleId, rolloff)

Set sample rolloff factor. Argument rolloff must be in range [0 .. 1]. Default value is 1. Argument sampleId obtained from the loadSample function.

setSampleRefDistance(samleId, distance)

Set distance at which sound was recorded. Argument sampleId obtained from the loadSample function.

setSampleCone(sampleId, outerGain, innerAngle, outerAngle)

Sets parameters of sound cone. Argument sampleId obtained from loadSample function. Each sample has the following arguments:

outerGain - the factor with which the sample gain is multiplied to determine the effective gain outside the cone defined by the outer angle. Must be in the range [0 .. 1], default is 0.

innerAngle – inside angle of the sound corner in degrees. Default is 360.

outerAngle – outer angle of the sound cone in degrees. Default is 360. When both inner and outer angle equals to 360 then the zone for angle depended attenuation is zero.

getSampleCone(sampleId)

Returns 3 numbers: outerGain, innerAngle and outerAngle. Argument sampleId obtained from loadSample function. Each sample has the following arguments:

outerGain - the factor with which the sample gain is multiplied to determine the effective gain outside the cone defined by the outer angle. Must be in the range [0 .. 1], default is 0.

innerAngle – inside angle of the sound corner in degrees. Default is 360.

outerAngle – outer angle of the sound cone in degrees. Default is 360. When both inner and outer angle equals to 360 then the zone for angle depended attenuation is zero.

setSampleRelative(sampleId, relative)

Marks the position of the sample relative to the camera. Argument sampleId obtained from the loadSample function. If the argument relative equals to 0, the sample position will be interpreted as the position relative to the aircraft reference point. If the argument relative equals to 1, the sample position will be interpreted as the position relative camera.

getSampleRelative(sampleId)

Returns 1 if the sample position interpreted relative camera position or 0 if sample position interpreted relative aircraft reference point. Argument sampleId obtained from loadSample function.

X-Plane Scenery

reloadScenery()

Reload X-Plane scenery files.

worldToLocal(latitude, longitude, altitude)

Converts X-Plane world coordinates to local coordinates. Returns local coordinates as 3 values: x, y and z.

localToWorld(x, y, z)

Converts X-Plane local coordinates to world coordinates. Returns world coordinates as 3 values: latitude, longitude and altitude.

probeTerrain(x, y, z)

Locates physical scenery mesh. Pass location of interest in local coordinate system. Returns query status and terrain parameters. Status may be one of following:

PROBE_HIT_TERRAIN – terrain found at specified location and more information available as additional return value: locationX, locationY, locationZ, normalX, normalY, normalZ, velocityX, velocityY, vlocityZ, isWet. Location is localtion of terrain point hit in local coordinates, normal is normal vector of terrain found, velocity is velocity vector of terrain found, isWet equals to true if sea found.

PROBE_ERROR – internal error.

PROBE_MISSED – terrain not found in specified location.

loadObject(fileName)

Loads X-Plane object from file. It looks for file using the same rules as for textures. Returns reference to loaded object or nil on errors.

unloadObject(object)

Unload object to free some memory.

drawObject(object, x, y, z, pitch, heading, roll, lighting, earthRelative)

Draw X-Plane object. X, y and z are local object coordinates. Pitch, heaading and roll are object orientation in degrees.

Lighting is a boolean; pass 1 to show the night version of object with night-only lights lit up. Pass 0 to show the daytime version of the object.

earthRelative controls the coordinate system. If this is 1, the rotations you specify are applied to the object after its coordinate system is transformed from local to earth-relative coordinates – that is, an object with no rotations will point toward true north and the Y axis will be up against gravity. If this is 0, the object is drawn with your rotations from local coordinates – that is, an object with no rotations is drawn pointing down the -Z axis and the Y axis of the object matches the local coordinate Y axis.

X-Plane Navigation

X-Plane supports number of different navigation points. Use the following constants to identify the navigation point type:

- NAV_UNKNOWN – unknown navigation point type
- NAV_AIRPORT – airfield or helipad
- NAV_NDB – non-directional beacon
- NAV_VOR – VOR site
- NAV_ILS – Instrument Landing System
- NAV_LOCALIZER – localizer part of ILS
- NAV_GLIDESLOPE – glide slope part of ILS
- NAV_OUTERMARKER – outer marker
- NAV_MIDDLEMARKER – middle marker
- NAV_INNERMARKER – inner marker
- NAV_FIX – intersection
- NAV_DME – distance measuring equipment
- NAV_LATLON – point defined by latitude and longitude
- NAV_NOT_FOUND – navigation point was not found. Returned by all functions when nothing to iterate

getFirstNavAid()

Returns descriptor of first entry in the navigation database. Use getNextNavAid to iterate all of the navigation points.

getNextNavAid(navRef)

Returns descriptor of next navigation aid in the sdatabase. Returns NAV_NOT_FOUND if no entries left in database.

findFirstNavAidOfType(type)

Returns reference of first navigation aid of specified type in database. See chapter introduction for list of navigation aids types. If no entries of specified type exists in database returns NAV_NOT_FOUND.

findLastNavAidOfType(type)

Returns reference of the last navaid of given type in database or NAV_NOT_FOUND if there are no navaids of that type in the database

findNavAid(nameFragment, idFragment, lat, lon, frequency, type)

Search in database for navaid. Argument type must be sum of required navaids type. Other arguments may equal nil if not needed. If lat and lot is not nil function returns reference to nearest navaid of specified type, otherwise it returns last found navaid.

getNavAidInfo(navRef)

Returns information about navaid as multiple values:

type, latitude, longitude, height, frequency, heading, ID, name, reg

All fields can equals to nil. All frequencies except NDB are multiplied by 100.

countFMSEntries()

Returns number of entries in FMS.

getDisplayedFMSEntry()

Returns index of entry displayed on FMS.

getDestinationFMSEntry()

Returns index of entry aircraft flying to.

setDisplayedFMSEntry(index)

Display FMS entry with specified index.

setDestinationFMSEntry(index)

This routine changes which entry the FMS is flying the aircraft toward.

getFMSEntryInfo(index)

Returns multiple values: type, ID, navRef, altitude, lat, lon. Each value may be nil. For lat/lon entry navRef equals to NAV_NOT_FOUND.

setFMSEntryInfo(index, navRef, altitude)

Change entry in FMS at specified index to navaid referenced by navRef argument. Can be used only for airports, fixes, VORs and NDBs.

setFMSEntryLatLon(index, lat, lon, altitude)

This routine changes the entry in the FMS to a lat/lon entry with the given coordinates.

clearFMSEntry(index)

This routine clears the given entry, potentially shortening the flight plan.

getGPSDestinationType()

Returns the type of the currently selected GPS destination, one of fix, airport, VOR or NDB.

getGPSDestination()

Returns reference to current GPS destination.

X-Plane Plugins Management

getMyID()

Returns ID of SASL plugin.

countPlugins()

Returns the total number of plug-ins that are loaded, both disabled and enabled.

getNthPlugin(index)

Returns the ID of a plug-in by index. Index is 0 based from 0 to countPlug-ins-1, inclusive. Plug-ins may be returned in any arbitrary order.

findPluginByPath(path)

Returns the plug-in ID of the plug-in whose file exists at the passed in absolute file system path. NO_PLUGIN_ID is returned if the path does not point to a currently loaded plug-in.

findPluginBySignature(signature)

Returns the plug-in ID of the plug-in whose signature matches what is passed in or NO_PLUGIN_ID if no running plug-in has this signature. Signatures are the best way to identify another plug-in since they are independent of the file system path of a plug-in or the human-readable plug-in name, and should be unique for all plug-ins.

getPluginInfo(pluginID)

Returns information about a plug-in as multiple values: name, filePath, signature, description. Name is the human-readable name of the plug-in. filePath - the absolute file path to the file that contains this plug-in. signature - a unique string that identifies this plug-in. description - a human-readable description of this plug-in.

isPluginEnabled(pluginID)

Returns whether the specified plug-in is enabled for running.

enablePlugin(pluginID)

Enables a plug-in if it is not already enabled. It returns 1 if the plug-in was enabled or successfully enables itself, 0 if it does not.

disablePlugin(pluginID)

Disables an enabled plug-in.