

X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)
 [Log in](#)

Category:XPLMPlugin

(Redirected from [XPLMPlugin](#))

[Category: Documentation](#)

[Documentation](#)

XPLMPlugin

These APIs provide facilities to find and work with other plugins and manage other plugins.

FINDING PLUGINS

These APIs allow you to find another plugin or yourself, or iterate across all plugins. For example, if you wrote an FMS plugin that needed to talk to an autopilot plugin, you could use these APIs to locate the autopilot plugin.

XPLMGetMyID

```
XPLM_API XPLMPluginID XPLMGetMyID(void);
```

This routine returns the plugin ID of the calling plug-in. Call this to get your own ID.

XPLMCountPlugins

```
XPLM_API int XPLMCountPlugins(void);
```

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

XPLMGetNthPlugin

```
XPLM_API XPLMPluginID XPLMGetNthPlugin(
    int inIndex);
```

This routine returns the ID of a plug-in by index. Index is 0 based from 0 to XPLMCountPlugins-1, inclusive. Plugins may be returned in any arbitrary order.

XPLMFindPluginByPath

```
XPLM_API XPLMPluginID XPLMFindPluginByPath(
    const char * inPath);
```

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

XPLMFindPluginBySignature

```
XPLM_API XPLMPluginID
XPLMFindPluginBySignature(
    const char * inSignature);
```

This routine returns the plug-in ID of the plug-in whose signature matches what is passed in or XPLM_NO_PLUGIN_ID if no running plug-in has this signature. Signatures are the best way to identify another plug-in as 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. Use this routine to locate another plugin that your plugin interoperates with

XPLMGetPluginInfo

```
XPLM_API void
XPLMGetPluginInfo(
    XPLMPluginID inPlugin,
    char * outName, /*
Can be NULL */
    char * outFilePath,
    char * outSignature,
    char *
outDescription); /* Can be NULL */
```

This routine returns information about a plug-in. Each parameter should be a pointer to a buffer of at least 256 characters, or NULL to not receive the information.

outName - the human-readable name of the plug-in. outFilePath - the absolute file path to the file that contains this plug-in. outSignature - a unique string that identifies this plug-in. outDescription - a human-readable description of this plug-in.

ENABLING/DISABLING PLUG-INS

These routines are used to work with plug-ins and manage them. Most plugins will not need to use these APIs.

XPLMIsPluginEnabled

```
XPLM_API int
XPLMIsPluginEnabled(
    XPLMPluginID inPluginID);
```

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

XPLMEnablePlugin

```
XPLM_API int
XPLMEnablePlugin(
    XPLMPluginID inPluginID);
```

This routine enables a plug-in if it is not already enabled. It returns 1 if the plugin was enabled or successfully enables itself, 0 if it does not. Plugins may fail to enable (for example, if resources cannot be acquired) by returning 0 from their XPluginEnable callback.

XPLMDisablePlugin

```
XPLM_API void XPLMDisablePlugin(
    XPLMPluginID inPluginID);
```

This routine disables an enabled plug-in.

XPLMReloadPlugins

```
XPLM_API void XPLMReloadPlugins(void);
```

This routine reloads all plug-ins. Once this routine is called and you return from the callback you were within (e.g. a menu select callback) you will receive your XPluginDisable and XPluginStop callbacks and your DLL will be unloaded, then the start process happens as if the sim was starting up.

INTERPLUGIN MESSAGING

Plugin messages are defined as 32-bit integers. Messages below 0x00FFFFFF are reserved for X-Plane and the plugin SDK.

Messages have two conceptual uses: notifications and commands. Commands are sent from one plugin to another to induce behavior; notifications are sent from one plugin to all others for informational purposes. It is important that commands and notifications not have the same values because this could cause a notification sent by one plugin to accidentally induce a command in another.

By convention, plugin-defined notifications should have the high bit set (e.g. be greater or equal to unsigned 0x8000000) while commands should have this bit be cleared.

The following messages are sent to your plugin by x-plane.

XPLM_MSG_PLANE_CRASHED

```
#define XPLM_MSG_PLANE_CRASHED 101
```

This message is sent to your plugin whenever the user's plane crashes.

XPLM_MSG_PLANE_LOADED

```
#define XPLM_MSG_PLANE_LOADED 102
```

This message is sent to your plugin whenever a new plane is loaded. The parameter is the number of the plane being loaded; 0 indicates the user's plane.

XPLM_MSG_AIRPORT_LOADED

```
#define XPLM_MSG_AIRPORT_LOADED 103
```

This messages is called whenever the user's plane is positioned at a new airport. The parameter is of type int, passed as the value of the pointer. (That is: the parameter is an int, not a pointer to an int.)

XPLM_MSG_SCENERY_LOADED

```
#define XPLM_MSG_SCENERY_LOADED 104
```

This message is sent whenever new scenery is loaded. Use datarefs to determine the new scenery files that were loaded.

XPLM_MSG_AIRPLANE_COUNT_CHANGED

```
#define XPLM_MSG_AIRPLANE_COUNT_CHANGED 105
```

This message is sent whenever the user adjusts the number of X-Plane aircraft models. You must use XPLMCountPlanes to find out how many planes are now available. This message will only be sent in XP7 and higher because in XP6 the number of aircraft is not user-adjustable.

Version 2.0

XPLM_MSG_PLANE_UNLOADED

```
#define XPLM_MSG_PLANE_UNLOADED 106
```

This message is sent to your plugin whenever a plane is unloaded. The parameter is the number of the plane being unloaded; 0 indicates the user's plane. The parameter is of type int, passed as the value of the pointer. (That is: the parameter is an int, not a pointer to an int.)

Version 2.1

XPLM_MSG_WILL_WRITE_PREFS

```
#define XPLM_MSG_WILL_WRITE_PREFS 107
```

This message is sent to your plugin right before X-Plane writes its preferences file. You can use this for two purposes: to write your own preferences, and to modify any datarefs to influence preferences output. For example, if your plugin temporarily modifies saved preferences, you can put them back to their default values here to avoid having the tweaks be persisted if your plugin is not loaded on the next invocation of X-Plane.

Version 2.1

XPLM_MSG_LIVERY_LOADED

```
#define XPLM_MSG_LIVERY_LOADED 108
```

This message is sent to your plugin right after a livery is loaded for an airplane. You can use this to check the new livery (via datarefs) and react accordingly. The parameter is of type int, passed as the value of a pointer and represents the aircraft plane number - 0 is the user's plane.

XPLMSendMessageToPlugin

```
XPLM_API void XPLMSendMessageToPlugin(
    XPLMPluginID inPlugin,
    int inMessage,
    void * inParam);
```

This function sends a message to another plug-in or X-Plane. Pass XPLM_NO_PLUGIN_ID to broadcast to all plug-ins. Only enabled plug-ins with a message receive function receive the message.

Version 2.0

Plugin Features API

The plugin features API allows your plugin to "sign up" for additional capabilities and plugin system features that are normally disabled for backward compatibility. This allows advanced plugins to "opt-in" to new behavior.

Each feature is defined by a permanent string name. The feature string names will vary with the particular installation of X-Plane, so plugins should not expect a feature to be guaranteed present.

XPLMFeatureEnumerator_f

```
typedef void (* XPLMFeatureEnumerator_f)(
    const char * inFeature,
    void * inRef);
```

You pass an XPLMFeatureEnumerator_f to get a list of all features supported by a given version running version of X-Plane. This routine is called once for each feature.

XPLMHasFeature

```
XPLM_API int XPLMHasFeature(
    const char * inFeature);
```

This returns 1 if the given installation of X-Plane supports a feature, or 0 if it does not.

XPLMIsFeatureEnabled

```
XPLM_API int XPLMIsFeatureEnabled(
    const char * inFeature);
```

This returns 1 if a feature is currently enabled for your plugin, or 0 if it is not enabled. It is an error to call this routine with an unsupported feature.

XPLMEnableFeature

```
XPLM_API void XPLMEnableFeature(
    const char * inFeature,
    int inEnable);
```

This routine enables or disables a feature for your plugin. This will change the running behavior of X-Plane and your plugin in some way, depending on the feature.

XPLMEnumerateFeatures

```
XPLM_API void XPLMEnumerateFeatures(
    XPLMFeatureEnumerator_f
    inEnumerator,
    void * inRef);
```

This routine calls your enumerator callback once for each feature that this running version of X-Plane supports. Use this routine to determine all of the features that X-Plane can support.

Pages in category "XPLMPlugin"

The following 24 pages are in this category, out of 24 total.

X

- [XPLM MSG AIRPLANE COUNT CHANGED](#)
- [XPLM MSG AIRPORT LOADED](#)
- [XPLM MSG LIVERY LOADED](#)
- [XPLM MSG PLANE CRASHED](#)
- [XPLM MSG PLANE LOADED](#)
- [XPLM MSG PLANE UNLOADED](#)
- [XPLM MSG SCENERY LOADED](#)
- [XPLM MSG WILL WRITE PREFS](#)

X cont.

- [XPLMCountPlugins](#)
- [XPLMDisablePlugin](#)
- [XPLMEnableFeature](#)
- [XPLMEnablePlugin](#)
- [XPLMEnumerateFeatures](#)
- [XPLMFeatureEnumerator f](#)
- [XPLMFindPluginByPath](#)
- [XPLMFindPluginBySignature](#)

X cont.

- [XPLMGetMyID](#)
- [XPLMGetNthPlugin](#)
- [XPLMGetPluginInfo](#)
- [XPLMHasFeature](#)
- [XPLMIsFeatureEnabled](#)
- [XPLMIsPluginEnabled](#)
- [XPLMReloadPlugins](#)
- [XPLMSendMessageToPlugin](#)

X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)

 [Log in](#)

Category:XPLMDataAccess

(Redirected from [XPLMDataAccess](#))

[Category: Documentation](#)

[Documentation](#)

XPLMDataAccess

XPLM Data Access API - Theory of Operation

The data access API gives you a generic, flexible, high performance way to read and write data to and from X-Plane and other plug-ins. For example, this API allows you to read and set the nav radios, get the plane location, determine the current effective graphics frame rate, etc.

The data access APIs are the way that you read and write data from the sim as well as other plugins.

The API works using opaque data references. A data reference is a source of data; you do not know where it comes from, but once you have it you can read the data quickly and possibly write it. To get a data reference, you look it up.

Data references are identified by verbose string names (sim/cockpit/radios/nav1_freq_hz). The actual numeric value of the data reference is implementation defined and is likely to change each time the simulator is run (or the plugin that provides the data reference is reloaded).

The task of looking up a data reference is relatively expensive; look up your data references once based on verbose strings, and save the opaque data reference value for the duration of your plugin's operation. Reading and writing data references is relatively fast (the cost is equivalent to two function calls through function pointers).

This allows data access to be high performance, while leaving in abstraction; since data references are opaque and are searched for, the underlying data access system can be rebuilt.

A note on typing: you must know the correct data type to read and write. APIs are provided for reading and writing data in a number of ways. You can also double check the data type for a data ref. Note that automatic conversion is not done for you.

A note for plugins sharing data with other plugins: the load order of plugins is not guaranteed. To make sure that every plugin publishing data has published their data references before other plugins try to subscribe, publish your data references in your start routine but resolve them the first time your 'enable' routine is called, or the first time they are needed in code.

X-Plane publishes well over 1000 datarefs; a complete list may be found in the reference section of the SDK online documentation (from the SDK home page, choose Documentation).

READING AND WRITING DATA

These routines allow you to access a wide variety of data from within x-plane and modify some of it.

XPLMDataRef

```
typedef void * XPLMDataRef;
```

A data ref is an opaque handle to data provided by the simulator or another plugin. It uniquely identifies one variable (or array of variables) over the lifetime of your plugin. You never hard code these values; you always get them from XPLMFindDataRef.

XPLMDataTypeID

This is an enumeration that defines the type of the data behind a data reference. This allows you to sanity check that the data type matches what you expect. But for the most part, you will know the type of data you are expecting from the online documentation.

Data types each take a bit field, so sets of data types may be formed.

xplmType_Unknown	0	Data of a type the current XPLM doesn't do.
xplmType_Int	1	A single 4-byte integer, native endian.
xplmType_Float	2	A single 4-byte float, native endian.
xplmType_Double	4	A single 8-byte double, native endian.
xplmType_FloatArray	8	An array of 4-byte floats, native endian.
xplmType_IntArray	16	An array of 4-byte integers, native endian.
xplmType_Data	32	A variable block of data.

XPLMFindDataRef

```
XPLM_API XPLMDataRef XPLMFindDataRef(
    const char *
    inDataRefName);
```

Given a c-style string that names the data ref, this routine looks up the actual opaque XPLMDataRef that you use to read and write the data. The string names for datarefs are published on the x-plane SDK web site.

This function returns NULL if the data ref cannot be found.

NOTE: this function is relatively expensive; save the XPLMDataRef this function returns for future use. Do not look up your data ref by string every time you need to read or write it.

XPLMCanWriteDataRef

```
XPLM_API int XPLMCanWriteDataRef(
    XPLMDataRef inDataRef);
```


Given a data ref, this routine returns true if you can successfully set the data, false otherwise. Some datarefs are read-only.

XPLMIsDataRefGood

```
XPLM_API int XPLMIsDataRefGood(
    XPLMDataRef inDataRef);
```

WARNING: This function is deprecated and should not be used. Datarefs are valid until plugins are reloaded or the sim quits. Plugins sharing datarefs should support these semantics by not unregistering datarefs during operation. (You should however unregister datarefs when your plugin is unloaded, as part of general resource cleanup.)

This function returns whether a data ref is still valid. If it returns false, you should refind the data ref from its original string. Calling an accessor function on a bad data ref will return a default value, typically 0 or 0-length data.

XPLMGetDataRefTypes

```
XPLM_API XPLMDataTypeID XPLMGetDataRefTypes(
    XPLMDataRef inDataRef);
```

This routine returns the types of the data ref for accessor use. If a data ref is available in multiple data types, they will all be returned.

DATA ACCESSORS

These routines read and write the data references. For each supported data type there is a reader and a writer.

If the data ref is invalid or the plugin that provides it is disabled or there is a type mismatch, the functions that read data will return 0 as a default value or not modify the passed in memory. The plugins that write data will not write under these circumstances or if the data ref is read-only. NOTE: to keep the overhead of reading datarefs low, these routines do not do full validation of a dataref; passing a junk value for a dataref can result in crashing the sim.

For array-style datarefs, you specify the number of items to read/write and the offset into the array; the actual number of items read or written is returned. This may be less to prevent an array-out-of-bounds error.

XPLMGetDatai

```
XPLM_API int XPLMGetDatai(
    XPLMDataRef inDataRef);
```

Read an integer data ref and return its value. The return value is the dataref value or 0 if the dataref is invalid/NULL or the plugin is disabled.

XPLMSetDatai

```
XPLM_API void XPLMSetDataI(
    XPLMDataRef inDataRef,
    int inValue);
```

Write a new value to an integer data ref. This routine is a no-op if the plugin publishing the dataref is disabled, the dataref is invalid, or the dataref is not writable.

XPLMGetDataf

```
XPLM_API float XPLMGetDataf(
    XPLMDataRef inDataRef);
```

Read a single precision floating point dataref and return its value. The return value is the dataref value or 0.0 if the dataref is invalid/NULL or the plugin is disabled.

XPLMSetDataf

```
XPLM_API void XPLMSetDataf(
    XPLMDataRef inDataRef,
    float inValue);
```

Write a new value to a single precision floating point data ref. This routine is a no-op if the plugin publishing the dataref is disabled, the dataref is invalid, or the dataref is not writable.

XPLMGetDataD

```
XPLM_API double XPLMGetDataD(
    XPLMDataRef inDataRef);
```

Read a double precision floating point dataref and return its value. The return value is the dataref value or 0.0 if the dataref is invalid/NULL or the plugin is disabled.

XPLMSetDataD

```
XPLM_API void XPLMSetDataD(
    XPLMDataRef inDataRef,
    double inValue);
```

Write a new value to a double precision floating point data ref. This routine is a no-op if the plugin publishing the dataref is disabled, the dataref is invalid, or the dataref is not writable.

XPLMGetDataVi

```
XPLM_API int XPLMGetDataVi(
    XPLMDataRef inDataRef,
    int *outValues,
    /* Can be NULL */
    int inOffset,
    int inMax);
```

Read a part of an integer array dataref. If you pass NULL for outVaules, the routine will return the size of the array, ignoring inOffset and inMax.

If outValues is not NULL, then up to inMax values are copied from the dataref into outValues, starting at inOffset in the dataref. If inMax + inOffset is larger than the size of the dataref, less than inMax values will be copied. The number of values copied is returned.

Note: the semantics of array datarefs are entirely implemented by the plugin (or X-Plane) that provides the dataref, not the SDK itself; the above description is how these datarefs are intended to work, but a rogue plugin may have different behavior.

XPLMSetDatavi

```
XPLM_API void XPLMSetDatavi(
    XPLMDataRef inDataRef,
    int * inValues,
    int inoffset,
    int inCount);
```

Write part or all of an integer array dataref. The values passed by inValues are written into the dataref starting at inOffset. Up to inCount values are written; however if the values would write "off the end" of the dataref array, then fewer values are written.

Note: the semantics of array datarefs are entirely implemented by the plugin (or X-Plane) that provides the dataref, not the SDK itself; the above description is how these datarefs are intended to work, but a rogue plugin may have different behavior.

XPLMGetDatavf

```
XPLM_API int XPLMGetDatavf(
    XPLMDataRef inDataRef,
    float * outValues,
    /* Can be NULL */
    int inOffset,
    int inMax);
```

Read a part of a single precision floating point array dataref. If you pass NULL for outVaules, the routine will return the size of the array, ignoring inOffset and inMax.

If outValues is not NULL, then up to inMax values are copied from the dataref into outValues, starting at inOffset in the dataref. If inMax + inOffset is larger than the size of the dataref, less than inMax values will be copied. The number of values copied is returned.

Note: the semantics of array datarefs are entirely implemented by the plugin (or X-Plane) that provides the dataref, not the SDK itself; the above description is how these datarefs are intended to work, but a rogue plugin may have different behavior.

XPLMSetDatavf

```
XPLM_API void XPLMSetDatavf(
    XPLMDataRef inDataRef,
    float * inValues,
    int inoffset,
```

```
int inCount);
```

Write part or all of a single precision floating point array dataref. The values passed by inValues are written into the dataref starting at inOffset. Up to inCount values are written; however if the values would write "off the end" of the dataref array, then fewer values are written.

Note: the semantics of array datarefs are entirely implemented by the plugin (or X-Plane) that provides the dataref, not the SDK itself; the above description is how these datarefs are intended to work, but a rogue plugin may have different behavior.

XPLMGetDataB

```
XPLM_API int XPLMGetDataB(
    XPLMDataRef inDataRef,
    void * outValue,
    int inOffset,
    int inMaxBytes);
```

Read a part of a byte array dataref. If you pass NULL for outVaules, the routine will return the size of the array, ignoring inOffset and inMax.

If outValues is not NULL, then up to inMax values are copied from the dataref into outValues, starting at inOffset in the dataref. If inMax + inOffset is larger than the size of the dataref, less than inMax values will be copied. The number of values copied is returned.

Note: the semantics of array datarefs are entirely implemented by the plugin (or X-Plane) that provides the dataref, not the SDK itself; the above description is how these datarefs are intended to work, but a rogue plugin may have different behavior.

XPLMSetDataB

```
XPLM_API void XPLMSetDataB(
    XPLMDataRef inDataRef,
    void * inValue,
    int inOffset,
    int inLength);
```

Write part or all of a byte array dataref. The values passed by inValues are written into the dataref starting at inOffset. Up to inCount values are written; however if the values would write "off the end" of the dataref array, then fewer values are written.

Note: the semantics of array datarefs are entirely implemented by the plugin (or X-Plane) that provides the dataref, not the SDK itself; the above description is how these datarefs are intended to work, but a rogue plugin may have different behavior.

PUBLISHING YOUR PLUGINS DATA

These functions allow you to create data references that other plug-ins can access via the above data access APIs. Data references published by other plugins operate the same as ones published by x-plane

in all manners except that your data reference will not be available to other plugins if/when your plugin is disabled.

You share data by registering data provider callback functions. When a plug-in requests your data, these callbacks are then called. You provide one callback to return the value when a plugin 'reads' it and another to change the value when a plugin 'writes' it.

Important: you must pick a prefix for your datarefs other than "sim/" - this prefix is reserved for X-Plane. The X-Plane SDK website contains a registry where authors can select a unique first word for dataref names, to prevent dataref collisions between plugins.

XPLMGetDatai_f

```
typedef int (* XPLMGetDatai_f)(
                                void *
                                inRefcon);
```

Data provider function pointers.

These define the function pointers you provide to get or set data. Note that you are passed a generic pointer for each one. This is the same pointer you pass in your register routine; you can use it to find global variables, etc.

The semantics of your callbacks are the same as the dataref accessor above - basically routines like XPLMGetDatai are just pass-throughs from a caller to your plugin. Be particularly mindful in implementing array dataref read-write accessors; you are responsible for avoiding overruns, supporting offset read/writes, and handling a read with a NULL buffer.

XPLMSetDataai_f

```
typedef void (* XPLMSetDataai_f)(
                                void *
                                inRefcon,
                                int
                                inValue);
```

XPLMGetDataf_f

```
typedef float (* XPLMGetDataf_f)(
                                void *
                                inRefcon);
```

XPLMSetDataf_f

```
typedef void (* XPLMSetDataf_f)(
                                void *
                                float
                                inRefcon,
                                inValue);
```

XPLMGetDatad_f

```
typedef double (* XPLMGetData_f)(
    void *
    inRefcon);
```

XPLMSetDatad_f

```
typedef void (* XPLMSetDatad_f)(
    void *
    double
    inRefcon,
    inValue);
```

XPLMGetDataavi_f

```
typedef int (* XPLMGetDataavi_f)(
    void *
    int *
    inRefcon,
    outValues,
    /* Can be NULL */
    int
    inOffset,
    inMax);
```

XPLMSetDataavi_f

```
typedef void (* XPLMSetDataavi_f)(
    void *
    int *
    int
    inRefcon,
    inValues,
    inOffset,
    inCount);
```

XPLMGetDatavf_f

```
typedef int (* XPLMGetDatavf_f)(
    void *
    float *
    inRefcon,
    outValues,
    /* Can be NULL */
    int
    inOffset,
    inMax);
```

XPLMSetDatavf_f

```
typedef void (* XPLMSetDatavf_f)(
    void *
    float *
    int
    inRefcon,
    inValues,
    inOffset,
    inCount);
```

XPLMGetDatab_f

```
typedef int (* XPLMGetDatab_f)(
    void *
    void *
    inRefcon,
    outValue,
    /* Can be NULL */
    int
    inOffset,
    inMaxLength);
```

XPLMSetDatab_f

```
typedef void (* XPLMSetDatab_f)(
    void *          inRefcon,
    void *          inValue,
    int             inOffset,
    int             inLength);
```

XPLMRegisterDataAccessor

```
XPLM_API XPLMDataRef XPLMRegisterDataAccessor(
    const char *      inDataName,
    XPLMDataTypeID    inDataType,
    int               inIsWritable,
    XPLMGetDatai_f    inReadInt,
    XPLMSetDataai_f    inWriteInt,
    XPLMGetDataf_f    inReadFloat,
    XPLMSetDataaf_f    inWriteFloat,
    XPLMGetDatad_f    inReadDouble,
    XPLMSetDatad_f    inWriteDouble,
    XPLMGetDataavi_f  inReadIntArray,
    XPLMSetDataavi_f  inWriteIntArray,
    XPLMGetDatavf_f    inReadFloatArray,
    XPLMSetDatavf_f    inWriteFloatArray,
    XPLMGetDataab_f    inReadData,
    XPLMSetDataab_f    inWriteData,
    void *            inReadRefcon,
    void *            inWriteRefcon);
```

This routine creates a new item of data that can be read and written. Pass in the data's full name for searching, the type(s) of the data for accessing, and whether the data can be written to. For each data type you support, pass in a read accessor function and a write accessor function if necessary. Pass NULL for data types you do not support or write accessors if you are read-only.

You are returned a data ref for the new item of data created. You can use this data ref to unregister your data later or read or write from it.

XPLMUnregisterDataAccessor

```
XPLM_API void XPLMUnregisterDataAccessor(
    XPLMDataRef    inDataRef);
```

Use this routine to unregister any data accessors you may have registered. You unregister a data ref by the XPLMDataRef you get back from registration. Once you unregister a data ref, your function pointer will not be called anymore.

For maximum compatibility, do not unregister your data accessors until final shutdown (when your XPluginStop routine is called). This allows other plugins to find your data reference once and use it for their entire time of operation.

SHARING DATA BETWEEN MULTIPLE PLUGINS

The data reference registration APIs from the previous section allow a plugin to publish data in a one-owner manner; the plugin that publishes the data reference owns the real memory that the data ref uses. This is satisfactory for most cases, but there are also cases where plugins need to share actual data.

With a shared data reference, no one plugin owns the actual memory for the data reference; the plugin SDK allocates that for you. When the first plugin asks to 'share' the data, the memory is allocated. When the data is changed, every plugin that is sharing the data is notified.

Shared data references differ from the 'owned' data references from the previous section in a few ways:

- With shared data references, any plugin can create the data reference; with owned plugins one plugin must create the data reference and others subscribe. (This can be a problem if you don't know which set of plugins will be present).
- With shared data references, every plugin that is sharing the data is notified when the data is changed. With owned data references, only the one owner is notified when the data is changed.
- With shared data references, you cannot access the physical memory of the data reference; you must use the XPLMGet... and XPLMSet... APIs. With an owned data reference, the one owning data reference can manipulate the data reference's memory in any way it sees fit.

Shared data references solve two problems: if you need to have a data reference used by several plugins but do not know which plugins will be installed, or if all plugins sharing data need to be notified when that data is changed, use shared data references.

XPLMDataChanged_f

```
typedef void (* XPLMDataChanged_f)(
                                void *                                inRefcon);
```

An XPLMDataChanged_f is a callback that the XPLM calls whenever any other plug-in modifies shared data. A refcon you provide is passed back to help identify which data is being changed. In response, you may want to call one of the XPLMGetDataxxx routines to find the new value of the data.

XPLMShareData

```
XPLM_API int                                XPLMShareData(
                                const char *                                inDataName,
                                XPLMDataTypeID                                inDataType,
                                XPLMDataChanged_f
                                void *
                                inNotificationFunc,
                                inNotificationRefcon);
```

This routine connects a plug-in to shared data, creating the shared data if necessary. inDataName is a standard path for the data ref, and inDataType specifies the type. This function will create the data if it does not exist. If the data already exists but the type does not match, an error is returned, so it is

important that plug-in authors collaborate to establish public standards for shared data.

If a notificationFunc is passed in and is not NULL, that notification function will be called whenever the data is modified. The notification refcon will be passed to it. This allows your plug-in to know which shared data was changed if multiple shared data are handled by one callback, or if the plug-in does not use global variables.

A one is returned for successfully creating or finding the shared data; a zero if the data already exists but is of the wrong type.

XPLMUnshareData

```
XPLM_API int XPLMUnshareData(
    const char * inDataName,
    XPLMDataTypeID inDataType,
    XPLMDataChanged_f
    inNotificationFunc,
    void * inNotificationRefcon);
```

This routine removes your notification function for shared data. Call it when done with the data to stop receiving change notifications. Arguments must match XPLMShareData. The actual memory will not necessarily be freed, since other plug-ins could be using it.

Pages in category "XPLMDataAccess"

The following 35 pages are in this category, out of 35 total.

X	X cont.	X cont.
■ XPLMCanWriteDataRef	■ XPLMGetDatai	■ XPLMSetDataf
■ XPLMDataChanged f	■ XPLMGetDatai f	■ XPLMSetDataf f
■ XPLMDataRef	■ XPLMGetDatavf	■ XPLMSetDatai
■ XPLMDataTypeID	■ XPLMGetDatavf f	■ XPLMSetDatai f
■ XPLMFindDataRef	■ XPLMGetDatavi	■ XPLMSetDatavf
■ XPLMGetDataRefTypes	■ XPLMGetDatavi f	■ XPLMSetDatavf f
■ XPLMGetDatab	■ XPLMIsDataRefGood	■ XPLMSetDatavi
■ XPLMGetDatab f	■ XPLMRegisterDataAccessor	■ XPLMSetDatavi f
■ XPLMGetDatad	■ XPLMSetDatab	■ XPLMShareData
■ XPLMGetDatad f	■ XPLMSetDatab f	■ XPLMUnregisterDataAccessor
■ XPLMGetDataf	■ XPLMSetDatad	■ XPLMUnshareData
■ XPLMGetDataf f	■ XPLMSetDatad f	



X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)

 [Log in](#)

Category:XPLMProcessing

(Redirected from [XPLMProcessing](#))

[Category: Documentation](#)
[Documentation](#)

XPLMProcessing

This API allows you to get regular callbacks during the flight loop, the part of X-Plane where the plane's position calculates the physics of flight, etc. Use these APIs to accomplish periodic tasks like logging data and performing I/O.

WARNING: Do NOT use these callbacks to draw! You cannot draw during flight loop callbacks. Use the drawing callbacks (see [XPLMDisplay](#) for more info) for graphics.

FLIGHT LOOP CALLBACKS

Version 2.1

XPLMFlightLoopPhaseType

You can register a flight loop callback to run either before or after the flight model is integrated by X-Plane.

Version 2.1

xplm_FlightLoop_Phase_BeforeFlightModel

0

Your callback runs before X-Plane integrates the flight model.

xplm_FlightLoop_Phase_AfterFlightModel

1

Your callback runs after X-Plane integrates the flight model.

Version 2.1

XPLMFlightLoopID

typedef void * XPLMFlightLoopID;

This is an opaque identifier for a flight loop callback. You can use this identifier to easily track and

remove your callbacks, or to use the new flight loop APIs.

XPLMFlightLoop_f

```
typedef float (* XPLMFlightLoop_f)(
    float
    inElapsedSinceLastCall,
    float
    inElapsedTimeSinceLastFlightLoop,
    int
    void *,
    inCounter,
    inRefcon);
```

This is your flight loop callback. Each time the flight loop is iterated through, you receive this call at the end. You receive a time since you were last called and a time since the last loop, as well as a loop counter. The 'phase' parameter is deprecated and should be ignored.

Your return value controls when you will next be called. Return 0 to stop receiving callbacks. Pass a positive number to specify how many seconds until the next callback. (You will be called at or after this time, not before.) Pass a negative number to specify how many loops must go by until you are called. For example, -1.0 means call me the very next loop. Try to run your flight loop as infrequently as is practical, and suspend it (using return value 0) when you do not need it; lots of flight loop callbacks that do nothing lowers x-plane's frame rate.

Your callback will NOT be unregistered if you return 0; it will merely be inactive.

The reference constant you passed to your loop is passed back to you.

Version 2.1

XPLMCreateFlightLoop_t

XPLMCreateFlightLoop_t contains the parameters to create a new flight loop callback. The structure can be expanded in future SDKs - always set structSize to the size of your structure in bytes.

```
typedef struct {
    int
    XPLMFlightLoopPhaseType
    XPLMFlightLoop_f
    void *
    structSize;
    phase;
    callbackFunc;
    refcon;
} XPLMCreateFlightLoop_t;
```

XPLMGetElapsedTime

```
XPLM_API float XPLMGetElapsedTime(void);
```

This routine returns the elapsed time since the sim started up in decimal seconds.

XPLMGetCycleNumber

```
XPLM_API int XPLMGetCycleNumber(void);
```

This routine returns a counter starting at zero for each sim cycle computed/video frame rendered.

XPLMRegisterFlightLoopCallback

```
XPLM_API void XPLMRegisterFlightLoopCallback(
    XPLMFlightLoop_f inFlightLoop,
    float inInterval,
    void * inRefcon);
```

This routine registers your flight loop callback. Pass in a pointer to a flight loop function and a refcon. inInterval defines when you will be called. Pass in a positive number to specify seconds from registration time to the next callback. Pass in a negative number to indicate when you will be called (e.g. pass -1 to be called at the next cycle). Pass 0 to not be called; your callback will be inactive.

XPLMUnregisterFlightLoopCallback

```
XPLM_API void XPLMUnregisterFlightLoopCallback(
    XPLMFlightLoop_f inFlightLoop,
    void * inRefcon);
```

This routine unregisters your flight loop callback. Do NOT call it from your flight loop callback. Once your flight loop callback is unregistered, it will not be called again.

XPLMSetFlightLoopCallbackInterval

```
XPLM_API void XPLMSetFlightLoopCallbackInterval(
    XPLMFlightLoop_f inFlightLoop,
    float inInterval,
    int inRelativeToNow,
    void * inRefcon);
```

This routine sets when a callback will be called. Do NOT call it from your callback; use the return value of the callback to change your callback interval from inside your callback.

inInterval is formatted the same way as in XPLMRegisterFlightLoopCallback; positive for seconds, negative for cycles, and 0 for deactivating the callback. If inRelativeToNow is 1, times are from the time of this call; otherwise they are from the time the callback was last called (or the time it was registered if it has never been called).

Version 2.1

XPLMCreateFlightLoop

```
XPLM_API XPLMFlightLoopID XPLMCreateFlightLoop(
    XPLMCreateFlightLoop_t * inParams);
```

This routine creates a flight loop callback and returns its ID. The flight loop callback is created using the input param struct, and is init'd to be unscheduled.

Version 2.1

XPLMDestroyFlightLoop

```
XPLM_API void XPLMDestroyFlightLoop(
    XPLMFlightLoopID
    inFlightLoopID);
```

This routine destroys a flight loop callback by ID.

Version 2.1

XPLMScheduleFlightLoop

```
XPLM_API void XPLMScheduleFlightLoop(
    XPLMFlightLoopID
    inFlightLoopID,
    float inInterval,
    int inRelativeToNow);
```

This routine schedules a flight loop callback for future execution. If `inInterval` is negative, it is run in a certain number of frames based on the absolute value of the input. If the interval is positive, it is a duration in seconds.

If `inRelativeToNow` is true, times are interpreted relative to the time this routine is called; otherwise they are relative to the last call time or the time the flight loop was registered (if never called).

THREAD SAFETY: it is legal to call this routine from any thread under the following conditions:

1. The call must be between the beginning of an `XPLMEnable` and the end of an `XPLMDisable` sequence. (That is, you must not call this routine from thread activity when your plugin was supposed to be disabled. Since plugins are only enabled while loaded, this also implies you cannot run this routine outside an `XPLMStart/XPLMStop` sequence.)
2. You may not call this routine re-entrantly for a single flight loop ID. (That is, you can't enable from multiple threads at the same time.)
3. You must call this routine between the time after `XPLMCreateFlightLoop` returns a value and the time you call `XPLMDestroyFlightLoop`. (That is, you must ensure that your threaded activity is within the life of the object. The SDK does not check this for you, nor does it synchronize destruction of the object.)
4. The object must be unscheduled if this routine is to be called from a thread other than the main thread.

The following 10 pages are in this category, out of 10 total.

X	X cont.	X cont.
<ul style="list-style-type: none">■ XPLMCreateFlightLoop■ XPLMDestroyFlightLoop■ XPLMFlightLoop f■ XPLMGetCycleNumber	<ul style="list-style-type: none">■ XPLMGetElapsedTime■ XPLMRegisterFlightLoopCallback■ XPLMRemoveMenuItem■ XPLMScheduleFlightLoop	<ul style="list-style-type: none">■ XPLMSetFlightLoopCallbackInterval■ XPLMUnregisterFlightLoopCallback
Main page Community portal Current events Recent changes Random page Help	What links here Related changes Special pages Printable version Permanent link	

X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)
 [Log in](#)

Category:XPLMDisplay

(Redirected from [XPLMDisplay](#))

[Category: Documentation](#)

[Documentation](#)

XPLMDisplay

XPLM Display APIs - THEORY OF OPERATION

This API provides the basic hooks to draw in X-Plane and create user interface. All X-Plane drawing is done in OpenGL. The X-Plane plug-in manager takes care of properly setting up the OpenGL context and matrices. You do not decide when in your code's execution to draw; X-Plane tells you when it is ready to have your plugin draw.

X-Plane's drawing strategy is straightforward: every "frame" the screen is rendered by drawing the 3-d scene (dome, ground, objects, airplanes, etc.) and then drawing the cockpit on top of it. Alpha blending is used to overlay the cockpit over the world (and the gauges over the panel, etc.).

There are two ways you can draw: directly and in a window.

Direct drawing involves drawing to the screen before or after X-Plane finishes a phase of drawing. When you draw directly, you can specify whether x-plane is to complete this phase or not. This allows you to do three things: draw before x-plane does (under it), draw after x-plane does (over it), or draw instead of x-plane.

To draw directly, you register a callback and specify what phase you want to intercept. The plug-in manager will call you over and over to draw that phase.

Direct drawing allows you to override scenery, panels, or anything. Note that you cannot assume that you are the only plug-in drawing at this phase.

Window drawing provides slightly higher level functionality. With window drawing you create a window that takes up a portion of the screen. Window drawing is always two dimensional. Window drawing is front-to-back controlled; you can specify that you want your window to be brought on top, and other plug-ins may put their window on top of you. Window drawing also allows you to sign up for key presses and receive mouse clicks.

There are three ways to get keystrokes:

If you create a window, the window can take keyboard focus. It will then receive all keystrokes. If no window has focus, X-Plane receives keystrokes. Use this to implement typing in dialog boxes, etc. Only one window may have focus at a time; your window will be notified if it loses focus.

If you need to associate key strokes with commands/functions in your plug-in, use a hot key. A hot key is a key-specific callback. Hotkeys are sent based on virtual key strokes, so any key may be distinctly mapped with any modifiers. Hot keys can be remapped by other plug-ins. As a plug-in, you don't have to worry about what your hot key ends up mapped to; other plug-ins may provide a UI for remapping keystrokes. So hotkeys allow a user to resolve conflicts and customize keystrokes.

If you need low level access to the keystroke stream, install a key sniffer. Key sniffers can be installed above everything or right in front of the sim.

DRAWING CALLBACKS

Basic drawing callbacks, for low level intercepting of render loop. The purpose of drawing callbacks is to provide targeted additions or replacements to x-plane's graphics environment (for example, to add extra custom objects, or replace drawing of the AI aircraft). Do not assume that the drawing callbacks will be called in the order implied by the enumerations. Also do not assume that each drawing phase ends before another begins; they may be nested.

XPLMDrawingPhase

This constant indicates which part of drawing we are in. Drawing is done from the back to the front. We get a callback before or after each item. Metaphases provide access to the beginning and end of the 3d (scene) and 2d (cockpit) drawing in a manner that is independent of new phases added via x-plane implementation.

WARNING: As X-Plane's scenery evolves, some drawing phases may cease to exist and new ones may be invented. If you need a particularly specific use of these codes, consult Austin and/or be prepared to revise your code as X-Plane evolves.

<code>xplm_Phase_FirstScene</code>	0	This is the earliest point at which you can draw in 3-d.
<code>xplm_Phase_Terrain</code>	5	Drawing of land and water.
<code>xplm_Phase_Airports</code>	10	Drawing runways and other airport detail.
<code>xplm_Phase_Vectors</code>	15	Drawing roads, trails, trains, etc.
<code>xplm_Phase_Objects</code>	20	3-d objects (houses, smokestacks, etc.
<code>xplm_Phase_Airplanes</code>	25	External views of airplanes, both yours and the AI aircraft.
<code>xplm_Phase_LastScene</code>	30	This is the last point at which you can draw in 3-d.
<code>xplm_Phase_FirstCockpit</code>	35	This is the first phase where you can draw in 2-d.
<code>xplm_Phase_Panel</code>	40	The non-moving parts of the aircraft panel.
<code>xplm_Phase_Gauges</code>	45	The moving parts of the aircraft panel.
<code>xplm_Phase_Window</code>	50	Floating windows from plugins.
<code>xplm_Phase_LastCockpit</code>	55	The last change to draw in 2d.
<code>xplm_Phase_LocalMap3D</code> [Version 2.0]	100	3-d Drawing for the local map. Use regular OpenGL coordinates to draw in this phase.
<code>xplm_Phase_LocalMap2D</code> [Version 2.0]	101	2-d Drawing of text over the local map.
<code>xplm_Phase_LocalMapProfile</code> [Version 2.0]	102	Drawing of the side-profile view in the local map screen.

XPLMDrawCallback_f

```
typedef int (* XPLMDrawCallback_f)(
    XPLMDrawingPhase    inPhase,
    int                  inIsBefore,
    void *               inRefcon);
```

This is the prototype for a low level drawing callback. You are passed in the phase and whether it is before or after. If you are before the phase, return 1 to let x-plane draw or 0 to suppress x-plane drawing. If you are after the phase the return value is ignored.

Refcon is a unique value that you specify when registering the callback, allowing you to slip a pointer to your own data to the callback.

Upon entry the OpenGL context will be correctly set up for you and OpenGL will be in 'local' coordinates for 3d drawing and panel coordinates for 2d drawing. The OpenGL state (texturing, etc.) will be unknown.

XPLMKeySniffer_f

```
typedef int (* XPLMKeySniffer_f)(
    char                inChar,
    XPLMKeyFlags        inFlags,
    char                inVirtualKey,
    void *              inRefcon);
```

This is the prototype for a low level key-sniffing function. Window-based UI should not use this! The windowing system provides high-level mediated keyboard access. By comparison, the key sniffer provides low level keyboard access.

Key sniffers are provided to allow libraries to provide non-windowed user interaction. For example, the MUI library uses a key sniffer to do pop-up text entry.

inKey is the character pressed, inRefCon is a value you supply during registration. Return 1 to pass the key on to the next sniffer, the window mgr, x-plane, or whomever is down stream. Return 0 to consume the key.

Warning: this API declares virtual keys as a signed character; however the VKEY #define macros in XPLMDefs.h define the vkeys using unsigned values (that is 0x80 instead of -0x80). So you may need to cast the incoming vkey to an unsigned char to get correct comparisons in C.

XPLMRegisterDrawCallback

```
XPLM_API int XPLMRegisterDrawCallback(
    XPLMDrawCallback_f    inCallback,
    XPLMDrawingPhase      inPhase,
    int                    inWantsBefore,
    void *                 inRefcon);
```

This routine registers a low level drawing callback. Pass in the phase you want to be called for and whether you want to be called before or after. This routine returns 1 if the registration was successful, or

0 if the phase does not exist in this version of x-plane. You may register a callback multiple times for the same or different phases as long as the refcon is unique each time.

XPLMUnregisterDrawCallback

```
XPLM_API int XPLMUnregisterDrawCallback(
    XPLMDrawCallback_f inCallback,
    XPLMDrawingPhase inPhase,
    int inWantsBefore,
    void * inRefcon);
```

This routine unregisters a draw callback. You must unregister a callback for each time you register a callback if you have registered it multiple times with different refcons. The routine returns 1 if it can find the callback to unregister, 0 otherwise.

XPLMRegisterKeySniffer

```
XPLM_API int XPLMRegisterKeySniffer(
    XPLMKeySniffer_f inCallback,
    int inBeforeWindows,
    void * inRefcon);
```

This routine registers a key sniffing callback. You specify whether you want to sniff before the window system, or only sniff keys the window system does not consume. You should ALMOST ALWAYS sniff non-control keys after the window system. When the window system consumes a key, it is because the user has "focused" a window. Consuming the key or taking action based on the key will produce very weird results. Returns 1 if successful.

XPLMUnregisterKeySniffer

```
XPLM_API int XPLMUnregisterKeySniffer(
    XPLMKeySniffer_f inCallback,
    int inBeforeWindows,
    void * inRefcon);
```

This routine unregisters a key sniffer. You must unregister a key sniffer for every time you register one with the exact same signature. Returns 1 if successful.

WINDOW API

Window API, for higher level drawing with UI interaction.

Note: all 2-d (and thus all window drawing) is done in 'cockpit pixels'. Even when the OpenGL window contains more than 1024x768 pixels, the cockpit drawing is magnified so that only 1024x768 pixels are available.

XPLMMouseEvent

When the mouse is clicked, your mouse click routine is called repeatedly. It is first called with the mouse down message. It is then called zero or more times with the mouse-drag message, and finally it is called once with the mouse up message. All of these messages will be directed to the same window.

```
xplm_MouseDown 1
xplm_MouseDrag 2
xplm_MouseUp 3
```

Version 2.0

XPLMCursorStatus

XPLMCursorStatus describes how you would like X-Plane to manage the cursor. See XPLMHandleCursor_f for more info.

Version 2.0

```
xplm_CursorDefault 0 X-Plane manages the cursor normally, plugin does not affect the cursor.
xplm_CursorHidden 1 X-Plane hides the cursor.
xplm_CursorArrow 2 X-Plane shows the cursor as the default arrow.
xplm_CursorCustom 3 X-Plane shows the cursor but lets you select an OS cursor.
```

XPLMWindowID

```
typedef void * XPLMWindowID;
```

This is an opaque identifier for a window. You use it to control your window. When you create a window, you will specify callbacks to handle drawing and mouse interaction, etc.

XPLMDrawWindow_f

```
typedef void (* XPLMDrawWindow_f)(
                                XPLMWindowID      inWindowID,
                                void *              inRefcon);
```

This function handles drawing. You are passed in your window and its refcon. Draw the window. You can use XPLM functions to find the current dimensions of your window, etc. When this callback is called, the OpenGL context will be set properly for cockpit drawing. NOTE: Because you are drawing your window over a background, you can make a translucent window easily by simply not filling in your entire window's bounds.

XPLMHandleKey_f

```
typedef void (* XPLMHandleKey_f)(
                                XPLMWindowID      inWindowID,
                                char               inKey,
                                XPLMKeyFlags       inFlags,
                                char               inVirtualKey,
                                void *             inRefcon,
                                int                losingFocus);
```

This function is called when a key is pressed or keyboard focus is taken away from your window. If losingFocus is 1, you are losing the keyboard focus, otherwise a key was pressed and inKey contains its

character. You are also passed your window and a refcon. Warning: this API declares virtual keys as a signed character; however the VKEY #define macros in XPLMDefs.h define the vkeys using unsigned values (that is 0x80 instead of -0x80). So you may need to cast the incoming vkey to an unsigned char to get correct comparisons in C.

XPLMHandleMouseClicked_f

```
typedef int (* XPLMHandleMouseClicked_f)(
    XPLMWindowID      inWindowID,
    int                x,
    int                y,
    XPLMMouseStatus    inMouse,
    void *              inRefcon);
```

You receive this call when the mouse button is pressed down or released. Between then these two calls is a drag. You receive the x and y of the click, your window, and a refcon. Return 1 to consume the click, or 0 to pass it through.

WARNING: passing clicks through windows (as of this writing) causes mouse tracking problems in X-Plane; do not use this feature!

Version 2.0

XPLMHandleCursor_f

```
typedef XPLMCursorStatus (* XPLMHandleCursor_f)(
    XPLMWindowID      inWindowID,
    int                x,
    int                y,
    void *              inRefcon);
```

The SDK calls your cursor status callback when the mouse is over your plugin window. Return a cursor status code to indicate how you would like X-Plane to manage the cursor. If you return `xplm_CursorDefault`, the SDK will try lower-Z-order plugin windows, then let the sim manage the cursor.

Note: you should never show or hide the cursor yourself - these APIs are typically reference-counted and thus cannot safely and predictably be used by the SDK. Instead return one of `xplm_CursorHidden` to hide the cursor or `xplm_CursorArrow`/`xplm_CursorCustom` to show the cursor.

If you want to implement a custom cursor by drawing a cursor in OpenGL, use `xplm_CursorHidden` to hide the OS cursor and draw the cursor using a 2-d drawing callback (after `xplm_Phase_Window` is probably a good choice). If you want to use a custom OS-based cursor, use `xplm_CursorCustom` to ask X-Plane to show the cursor but not affect its image. You can then use an OS specific call like `SetThemeCursor` (Mac) or `SetCursor/LoadCursor` (Windows).

Version 2.0

XPLMHandleMouseWheel_f

```
typedef int (* XPLMHandleMouseWheel_f)(
    XPLMWindowID    inWindowID,
    int             x,
    int             y,
    int             wheel,
    int             clicks,
    void *          inRefcon);
```

The SDK calls your mouse wheel callback when one of the mouse wheels is turned within your window. Return 1 to consume the mouse wheel clicks or 0 to pass them on to a lower window. (You should consume mouse wheel clicks even if they do nothing if your window appears opaque to the user.) The number of clicks indicates how far the wheel was turned since the last callback. The wheel is 0 for the vertical axis or 1 for the horizontal axis (for OS/mouse combinations that support this).

Version 2.0

XPLMCreateWindow_t

The XPLMCreateWindow_t structure defines all of the parameters used to create a window using XPLMCreateWindowEx. The structure will be expanded in future SDK APIs to include more features. Always set the structSize member to the size of your struct in bytes!

```
typedef struct {
    int             structSize;
    int             left;
    int             top;
    int             right;
    int             bottom;
    int             visible;
    XPLMDrawWindow_f drawWindowFunc;
    XPLMHandleMouseClicked_f handleMouseClickedFunc;
    XPLMHandleKey_f handleKeyFunc;
    XPLMHandleCursor_f handleCursorFunc;
    XPLMHandleMouseWheel_f handleMouseWheelFunc;
    void *          refcon;
} XPLMCreateWindow_t;
```

XPLMGetScreenSize

```
XPLM_API void XPLMGetScreenSize(
    /* Can be NULL */ int * outWidth,
    /* Can be NULL */ int * outHeight);
```

This routine returns the size of the size of the X-Plane OpenGL window in pixels. Please note that this is not the size of the screen when doing 2-d drawing (the 2-d screen is currently always 1024x768, and graphics are scaled up by OpenGL when doing 2-d drawing for higher-res monitors). This number can be used to get a rough idea of the amount of detail the user will be able to see when drawing in 3-d.

XPLMGetMouseLocation

```
XPLM_API void XPLMGetMouseLocation(
    int * outX, /*
```

```

Can be NULL */
int * outY); /*
Can be NULL */

```

This routine returns the current mouse location in cockpit pixels. The bottom left corner of the display is 0,0. Pass NULL to not receive info about either parameter.

XPLMCreateWindow

```

XPLM_API XPLMWindowID XPLMCreateWindow(
    int inLeft,
    int inTop,
    int inRight,
    int inBottom,
    int isVisible,
    XPLMDrawWindow_f inDrawCallback,
    XPLMHandleKey_f inKeyCallback,
    XPLMHandleMouseClicked_f
    inMouseCallback,
    void * inRefcon);

```

This routine creates a new window. Pass in the dimensions and offsets to the window's bottom left corner from the bottom left of the screen. You can specify whether the window is initially visible or not. Also, you pass in three callbacks to run the window and a refcon. This function returns a window ID you can use to refer to the new window.

NOTE: windows do not have "frames"; you are responsible for drawing the background and frame of the window. Higher level libraries have routines which make this easy.

Version 2.0

XPLMCreateWindowEx

```

XPLM_API XPLMWindowID XPLMCreateWindowEx(
    XPLMCreateWindow_t * inParams);

```

This routine creates a new window - you pass in an XPLMCreateWindow_t structure with all of the fields set in. You must set the structSize of the structure to the size of the actual structure you used. Also, you must provide funtions for every callback - you may not leave them null! (If you do not support the cursor or mouse wheel, use functions that return the default values.) The numeric values of the XPMCreateWindow_t structure correspond to the parameters of XPLMCreateWindow.

XPLMDestroyWindow

```

XPLM_API void XPLMDestroyWindow(
    XPLMWindowID inWindowID);

```

This routine destroys a window. The callbacks are not called after this call. Keyboard focus is removed from the window before destroying it.

XPLMGetWindowGeometry

```

XPLM_API void XPLMGetWindowGeometry(
    XPLMWindowID inWindowID,
    int * outLeft, /*
    Can be NULL */
    int * outTop, /*
    Can be NULL */
    int * outRight,
    /* Can be NULL */
    int * outBottom);

```

This routine returns the position and size of a window in cockpit pixels. Pass NULL to not receive any paramter.

XPLMSetWindowGeometry

```

XPLM_API void XPLMSetWindowGeometry(
    XPLMWindowID inWindowID,
    int inLeft,
    int inTop,
    int inRight,
    int inBottom);

```

This routine allows you to set the position or height aspects of a window.

XPLMGetWindowIsVisible

```

XPLM_API int XPLMGetWindowIsVisible(
    XPLMWindowID inWindowID);

```

This routine returns whether a window is visible.

XPLMSetWindowIsVisible

```

XPLM_API void XPLMSetWindowIsVisible(
    XPLMWindowID inWindowID,
    int inIsVisible);

```

This routine shows or hides a window.

XPLMGetWindowRefCon

```

XPLM_API void * XPLMGetWindowRefCon(
    XPLMWindowID inWindowID);

```

This routine returns a windows refcon, the unique value you can use for your own purposes.

XPLMSetWindowRefCon

```

XPLM_API void XPLMSetWindowRefCon(
    XPLMWindowID inWindowID,
    void * inRefcon);

```

This routine sets a window's reference constant. Use this to pass data to yourself in the callbacks.

XPLMTakeKeyboardFocus

```
XPLM_API void XPLMTakeKeyboardFocus(
    XPLMWindowID inWindow);
```

This routine gives a specific window keyboard focus. Keystrokes will be sent to that window. Pass a window ID of 0 to pass keyboard strokes directly to X-Plane.

XPLMBringWindowToFront

```
XPLM_API void XPLMBringWindowToFront(
    XPLMWindowID inWindow);
```

This routine brings the window to the front of the Z-order. Windows are brought to the front when they are created...beyond that you should make sure you are front before handling mouse clicks.

XPLMIsWindowInFront

```
XPLM_API int XPLMIsWindowInFront(
    XPLMWindowID inWindow);
```

This routine returns true if you pass in the ID of the frontmost visible window.

HOT KEYS

Hot Keys - keystrokes that can be managed by others.

XPLMHotKey_f

```
typedef void (* XPLMHotKey_f)(
    void * inRefcon);
```

Your hot key callback simply takes a pointer of your choosing.

XPLMHotKeyID

```
typedef void * XPLMHotKeyID;
```

Hot keys are identified by opaque IDs.

XPLMRegisterHotKey

```
XPLM_API XPLMHotKeyID XPLMRegisterHotKey(
    char inVirtualKey,
    XPLMKeyFlags inFlags,
    const char * inDescription,
    XPLMHotKey_f inCallback,
    void * inRefcon);
```

This routine registers a hot key. You specify your preferred key stroke virtual key/flag combination, a description of what your callback does (so other plug-ins can describe the plug-in to the user for

remapping) and a callback function and opaque pointer to pass in). A new hot key ID is returned. During execution, the actual key associated with your hot key may change, but you are insulated from this.

XPLMUnregisterHotKey

```
XPLM_API void XPLMUnregisterHotKey(
    XPLMHotKeyID inHotKey);
```

This API unregisters a hot key. You can only register your own hot keys.

XPLMCountHotKeys

```
XPLM_API int XPLMCountHotKeys(void);
```

Returns the number of current hot keys.

XPLMGetNthHotKey

```
XPLM_API XPLMHotKeyID XPLMGetNthHotKey(
    int inIndex);
```

Returns a hot key by index, for iteration on all hot keys.

XPLMGetHotKeyInfo

```
XPLM_API void XPLMGetHotKeyInfo(
    XPLMHotKeyID inHotKey,
    char * outVirtualKey,
    /* Can be NULL */
    XPLMKeyFlags * outFlags, /*
    Can be NULL */
    char *
    outDescription, /* Can be NULL */
    XPLMPluginID * outPlugin);
/* Can be NULL */
```

Returns information about the hot key. Return NULL for any parameter you don't want info about. The description should be at least 512 chars long.

XPLMSetHotKeyCombination

```
XPLM_API void XPLMSetHotKeyCombination(
    XPLMHotKeyID inHotKey,
    char inVirtualKey,
    XPLMKeyFlags inFlags);
```

XPLMSetHotKeyCombination remaps a hot keys keystrokes. You may remap another plugin's keystrokes.

Pages in category "XPLMDisplay"

The following 38 pages are in this category, out of 38 total.

X

X cont.

X cont.

- [XPLMBringWindowToFront](#)
 - [XPLMCountHotKeys](#)
 - [XPLMCreateWindow](#)
 - [XPLMCreateWindow t](#)
 - [XPLMCreateWindowEx](#)
 - [XPLMCursorStatus](#)
 - [XPLMDestroyWindow](#)
 - [XPLMDrawCallback f](#)
 - [XPLMDrawWindow f](#)
 - [XPLMDrawingPhase](#)
 - [XPLMGetHotKeyInfo](#)
 - [XPLMGetMouseLocation](#)
 - [XPLMGetNthHotKey](#)
- [XPLMGetScreenSize](#)
 - [XPLMGetWindowGeometry](#)
 - [XPLMGetWindowsVisible](#)
 - [XPLMGetWindowRefCon](#)
 - [XPLMHandleCursor f](#)
 - [XPLMHandleKey f](#)
 - [XPLMHandleMouseClick f](#)
 - [XPLMHandleMouseWheel f](#)
 - [XPLMHotKey f](#)
 - [XPLMHotKeyID](#)
 - [XPLMIsWindowInFront](#)
 - [XPLMKeySniffer f](#)
 - [XPLMMouseStatus](#)
- [XPLMRegisterDrawCallback](#)
 - [XPLMRegisterHotKey](#)
 - [XPLMRegisterKeySniffer](#)
 - [XPLMSetHotKeyCombination](#)
 - [XPLMSetWindowGeometry](#)
 - [XPLMSetWindowsVisible](#)
 - [XPLMSetWindowRefCon](#)
 - [XPLMTakeKeyboardFocus](#)
 - [XPLMUnregisterDrawCallback](#)
 - [XPLMUnregisterHotKey](#)
 - [XPLMUnregisterKeySniffer](#)
 - [XPLMWindowID](#)

X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)
 [Log in](#)

Category:XPLMMenu

(Redirected from [XPLMMenu](#))

[Category: Documentation](#)

[Documentation](#)

XPLMMenu

XPLMMenu - Theory of Operation

Plug-ins can create menus in the menu bar of X-Plane. This is done by creating a menu and then creating items. Menus are referred to by an opaque ID. Items are referred to by index number. For each menu and item you specify a void *. Per menu you specify a handler function that is called with each void * when the menu item is picked. Menu item indices are zero based.

XPLM MENU

XPLMMenuCheck

These enumerations define the various 'check' states for an X-Plane menu. 'checking' in x-plane actually appears as a light which may or may not be lit. So there are three possible states.

```
xplm_Menu_NoCheck    0 there is no symbol to the left of the menu item.
xplm_Menu_Unchecked  1 the menu has a mark next to it that is unmarked (not lit).
xplm_Menu_Checked    2 the menu has a mark next to it that is checked (lit).
```

XPLMMenuID

```
typedef void * XPLMMenuID;
```

This is a unique ID for each menu you create.

XPLMMenuHandler_f

```
typedef void (* XPLMMenuHandler_f)(
    void *          inMenuRef,
    void *          inItemRef);
```

A menu handler function takes two reference pointers, one for the menu (specified when the menu was created) and one for the item (specified when the item was created).

XPLMFindPluginsMenu

```
XPLM_API XPLMMenuID XPLMFindPluginsMenu(void);
```

This function returns the ID of the plug-ins menu, which is created for you at startup.

XPLMCreateMenu

```
XPLM_API XPLMMenuID XPLMCreateMenu(
    const char *      inName,
    XPLMMenuID        inParentMenu,
    int                inParentItem,
    XPLMMenuHandler_f inHandler,
    void *             inMenuRef);
```

This function creates a new menu and returns its ID. It returns NULL if the menu cannot be created. Pass in a parent menu ID and an item index to create a submenu, or NULL for the parent menu to put the menu in the menu bar. The menu's name is only used if the menu is in the menubar. You also pass a handler function and a menu reference value. Pass NULL for the handler if you do not need callbacks from the menu (for example, if it only contains submenus).

Important: you must pass a valid, non-empty menu title even if the menu is a submenu where the title is not visible.

XPLMDestroyMenu

```
XPLM_API void XPLMDestroyMenu(
    XPLMMenuID inMenuID);
```

This function destroys a menu that you have created. Use this to remove a submenu if necessary. (Normally this function will not be necessary.)

XPLMClearAllMenuItems

```
XPLM_API void XPLMClearAllMenuItems(
    XPLMMenuID inMenuID);
```

This function removes all menu items from a menu, allowing you to rebuild it. Use this function if you need to change the number of items on a menu.

XPLMAppendMenuItem

```
XPLM_API int XPLMAppendMenuItem(
    XPLMMenuID inMenu,
    const char * inItemName,
    void *      inItemRef,
    int         inForceEnglish);
```

This routine appends a new menu item to the bottom of a menu and returns its index. Pass in the menu to add the item to, the item's name, and a void * ref for this item. If you pass in inForceEnglish, this menu item will be drawn using the english character set no matter what language x-plane is running in. Otherwise the menu item will be drawn localized. (An example of why you'd want to do this is for a proper name.) See XPLMUtilities for determining the current language.

XPLMAppendMenuSeparator

```
XPLM_API void XPLMAppendMenuSeparator(
    XPLMMenuID inMenu);
```

This routine adds a separator to the end of a menu.

XPLMSetMenuItemName

```
XPLM_API void XPLMSetMenuItemName(
    XPLMMenuID inMenu,
    int inIndex,
    const char * inItemName,
    int inForceEnglish);
```

This routine changes the name of an existing menu item. Pass in the menu ID and the index of the menu item.

XPLMCheckMenuItem

```
XPLM_API void XPLMCheckMenuItem(
    XPLMMenuID inMenu,
    int index,
    XPLMMenuCheck inCheck);
```

Set whether a menu item is checked. Pass in the menu ID and item index.

XPLMCheckMenuItemState

```
XPLM_API void XPLMCheckMenuItemState(
    XPLMMenuID inMenu,
    int index,
    XPLMMenuCheck * outCheck);
```

This routine returns whether a menu item is checked or not. A menu item's check mark may be on or off, or a menu may not have an icon at all.

XPLMEnableMenuItem

```
XPLM_API void XPLMEnableMenuItem(
    XPLMMenuID inMenu,
    int index,
    int enabled);
```

Sets whether this menu item is enabled. Items start out enabled.

Version 2.1

XPLMRemoveMenuItem

```
XPLM_API void XPLMRemoveMenuItem(
```

```
XPLMMenuID  
int  
  
inMenu,  
inIndex);
```

Removes one item from a menu. Note that all menu items below are moved up one; your plugin must track the change in index numbers.

Pages in category "XPLMMenus"

The following 13 pages are in this category, out of 13 total.

X

- [XPLMAppendMenuItem](#)
- [XPLMAppendMenuSeparator](#)
- [XPLMCheckMenuItem](#)
- [XPLMCheckMenuItemState](#)
- [XPLMClearAllMenuItems](#)

X cont.

- [XPLMCreateMenu](#)
- [XPLMDestroyMenu](#)
- [XPLMEnableMenuItem](#)
- [XPLMFindPluginsMenu](#)
- [XPLMMenuCheck](#)

X cont.

- [XPLMMenuHandler f](#)
- [XPLMMenuID](#)
- [XPLMSetMenuItemName](#)



X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)

 [Log in](#)

Category:XPLMGraphics

(Redirected from [XPLMGraphics](#))

[Category: Documentation](#)

[Documentation](#)

XPLMGraphics

Graphics routines for X-Plane and OpenGL.

A few notes on coordinate systems:

X-Plane uses three kinds of coordinates. Global coordinates are specified as latitude, longitude and elevation. This coordinate system never changes but is not very precise.

OpenGL (or 'local') coordinates are cartesian and shift with the plane. They offer more precision and are used for 3-d OpenGL drawing. The X axis is aligned east-west with positive X meaning east. The Y axis is aligned straight up and down at the point 0,0,0 (but since the earth is round it is not truly straight up and down at other points). The Z axis is aligned north-south at 0, 0, 0 with positive Z pointing south (but since the earth is round it isn't exactly north-south as you move east or west of 0, 0, 0). One unit is one meter and the point 0,0,0 is on the surface of the earth at sea level for some latitude and longitude picked by the sim such that the user's aircraft is reasonably nearby.

Cockpit coordinates are 2d, with the X axis horizontal and the Y axis vertical. The point 0,0 is the bottom left and 1024,768 is the upper right of the screen. This is true no matter what resolution the user's monitor is in; when running in higher resolution, graphics will be scaled.

Use X-Plane's routines to convert between global and local coordinates. Do not attempt to do this conversion yourself; the precise 'roundness' of X-Plane's physics model may not match your own, and (to make things weirder) the user can potentially customize the physics of the current planet.

X-PLANE GRAPHICS

These routines allow you to use OpenGL with X-Plane.

XPLMTextureID

XPLM Texture IDs name well-known textures in the sim for you to use. This allows you to recycle textures from X-Plane, saving VRAM.

- | | | |
|---|---|--|
| xplm_Tex_GeneralInterface | 0 | The bitmap that contains window outlines, button outlines, fonts, etc. |
| xplm_Tex_AircraftPaint | 1 | The exterior paint for the user's aircraft (daytime). |
| xplm_Tex_AircraftLiteMap | 2 | The exterior light map for the user's aircraft. |

XPLMSetGraphicsState

```
XPLM_API void XPLMSetGraphicsState(
    int inEnableFog,
    int inNumberTexUnits,
    int inEnableLighting,
    int inEnableAlphaTesting,
    int inEnableAlphaBlending,
    int inEnableDepthTesting,
    int inEnableDepthWriting);
```

XPLMSetGraphicsState changes OpenGL's graphics state in a number of ways:

inEnableFog - enables or disables fog, equivalent to: glEnable(GL_FOG);

inNumberTexUnits - enables or disables a number of multitexturing units. If the number is 0, 2d texturing is disabled entirely, as in glDisable(GL_TEXTURE_2D); Otherwise, 2d texturing is enabled, and a number of multitexturing units are enabled sequentially, starting with unit 0, e.g. glActiveTextureARB(GL_TEXTURE0_ARB); glEnable (GL_TEXTURE_2D);

inEnableLighting - enables or disables OpenGL lighting, e.g. glEnable(GL_LIGHTING); glEnable(GL_LIGHT0);

inEnableAlphaTesting - enables or disables the alpha test per pixel, e.g. glEnable(GL_ALPHA_TEST);

inEnableAlphaBlending - enables or disables alpha blending per pixel, e.g. glEnable(GL_BLEND);

inEnableDepthTesting - enables per pixel depth testing, as in glEnable(GL_DEPTH_TEST);

inEnableDepthWriting - enables writing back of depth information to the depth buffer, as in glDepthMask(GL_TRUE);

The purpose of this function is to change OpenGL state while keeping X-Plane aware of the state changes; this keeps X-Plane from getting surprised by OGL state changes, and prevents X-Plane and plug-ins from having to set all state before all draws; XPLMSetGraphicsState internally skips calls to change state that is already properly enabled.

X-Plane does not have a 'default' OGL state to plug-ins; plug-ins should totally set OGL state before drawing. Use XPLMSetGraphicsState instead of any of the above OpenGL calls.

WARNING: Any routine that performs drawing (e.g. XPLMDrawString or widget code) may change X-Plane's state. Always set state before drawing after unknown code has executed.

XPLMBindTexture2d

```
XPLM_API void XPLMBindTexture2d(
```



```

                                int          inTextureNum,
                                int
    inTextureUnit);

```

XPLMBindTexture2d changes what texture is bound to the 2d texturing target. This routine caches the current 2d texture across all texturing units in the sim and plug-ins, preventing extraneous binding. For example, consider several plug-ins running in series; if they all use the 'general interface' bitmap to do UI, calling this function will skip the rebinding of the general interface texture on all but the first plug-in, which can provide better frame rate son some graphics cards.

inTextureID is the ID of the texture object to bind; inTextureUnit is a zero-based texture unit (e.g. 0 for the first one), up to a maximum of 4 units. (This number may increase in future versions of x-plane.)

Use this routine instead of glBindTexture(GL_TEXTURE_2D,);

XPLMGenerateTextureNumbers

```

XPLM_API void          XPLMGenerateTextureNumbers(
                        int *          outTextureIDs,
                        int          inCount);

```

This routine generates unused texture numbers that a plug-in can use to safely bind textures. Use this routine instead of glGenTextures; glGenTextures will allocate texture numbers in ranges that X-Plane reserves for its own use but does not always use; for example, it might provide an ID within the range of textures reserved for terrain...loading a new .env file as the plane flies might then cause X-Plane to use this texture ID. X-Plane will then overwrite the plug-ins texture. This routine returns texture IDs that are out of X-Plane's usage range.

XPLMGetTexture

```

XPLM_API int          XPLMGetTexture(
                        XPLMTextureID          inTexture);

```

XPLMGetTexture returns the OpenGL texture enumeration of an X-Plane texture based on a generic identifying code. For example, you can get the texture for X-Plane's UI bitmaps. This allows you to build new gauges that take advantage of x-plane's textures, for smooth artwork integration and also saving texture memory. Note that the texture might not be loaded yet, depending on what the plane's panel contains.

OPEN ISSUE: We really need a way to make sure X-Plane loads this texture if it isn't around, or at least a way to find out whether it is loaded or not.

XPLMWorldToLocal

```

XPLM_API void          XPLMWorldToLocal(
                        double          inLatitude,
                        double          inLongitude,
                        double          inAltitude,
                        double *          outX,
                        double *          outY,
                        double *          outZ);

```

This routine translates coordinates from latitude, longitude, and altitude to local scene coordinates. Latitude and longitude are in decimal degrees, and altitude is in meters MSL (mean sea level). The XYZ coordinates are in meters in the local OpenGL coordinate system.

XPLMLocalToWorld

```
XPLM_API void XPLMLocalToWorld(
    double inX,
    double inY,
    double inZ,
    double * outLatitude,
    double * outLongitude,
    double * outAltitude);
```

This routine translates a local coordinate triplet back into latitude, longitude, and altitude. Latitude and longitude are in decimal degrees, and altitude is in meters MSL (mean sea level). The XYZ coordinates are in meters in the local OpenGL coordinate system.

NOTE: world coordinates are less precise than local coordinates; you should try to avoid round tripping from local to world and back.

XPLMDrawTranslucentDarkBox

```
XPLM_API void XPLMDrawTranslucentDarkBox(
    int inLeft,
    int inTop,
    int inRight,
    int inBottom);
```

This routine draws a translucent dark box, partially obscuring parts of the screen but making text easy to read. This is the same graphics primitive used by X-Plane to show text files and ATC info.

X-PLANE TEXT

XPLMFontID

X-Plane features some fixed-character fonts. Each font may have its own metrics.

WARNING: Some of these fonts are no longer supported or may have changed geometries. For maximum compatibility, see the comments below.

Note: X-Plane 7 supports proportional-spaced fonts. Since no measuring routine is available yet, the SDK will normally draw using a fixed-width font. You can use a dataref to enable proportional font drawing on XP7 if you want to.

xplmFont_Basic	0	Mono-spaced font for user interface. Available in all versions of the SDK.
xplmFont_Menus	1	Deprecated, do not use.

xplmFont_Metal	2	Deprecated, do not use.
xplmFont_Led	3	Deprecated, do not use.
xplmFont_LedWide	4	Deprecated, do not use.
xplmFont_PanelHUD	5	Deprecated, do not use.
xplmFont_PanelEFIS	6	Deprecated, do not use.
xplmFont_PanelGPS	7	Deprecated, do not use.
xplmFont_RadiosGA	8	Deprecated, do not use.
xplmFont_RadiosBC	9	Deprecated, do not use.
xplmFont_RadiosHM	10	Deprecated, do not use.
xplmFont_RadiosGANarrow	11	Deprecated, do not use.
xplmFont_RadiosBCNarrow	12	Deprecated, do not use.
xplmFont_RadiosHMNarrow	13	Deprecated, do not use.
xplmFont_Timer	14	Deprecated, do not use.
xplmFont_FullRound	15	Deprecated, do not use.
xplmFont_SmallRound	16	Deprecated, do not use.
xplmFont_Menus_Localized	17	Deprecated, do not use.
xplmFont_Proportional [Version 2.0]	18	Proportional UI font.

XPLMDrawString

```
XPLM_API void XPLMDrawString(
    float *      inColorRGB,
    int          inXOffset,
    int          inYOffset,
    char *       inChar,
    int *        inWordWrapWidth, /* Can be NULL */
    XPLMFontID  inFontID);
```

This routine draws a NULL terminated string in a given font. Pass in the lower left pixel that the character is to be drawn onto. Also pass the character and font ID. This function returns the x offset plus the width of all drawn characters. The color to draw in is specified as a pointer to an array of three floating point colors, representing RGB intensities from 0.0 to 1.0.

XPLMDrawNumber

```
XPLM_API void XPLMDrawNumber(
    float *      inColorRGB,
    int          inXOffset,
    int          inYOffset,
    double       inValue,
    int          inDigits,
    int          inDecimals,
    int          inShowSign,
    XPLMFontID  inFontID);
```

This routine draws a number similar to the digit editing fields in PlaneMaker and data output display in X-Plane. Pass in a color, a position, a floating point value, and formatting info. Specify how many integer and how many decimal digits to show and whether to show a sign, as well as a character set. This routine returns the xOffset plus width of the string drawn.

XPLMGetFontDimensions

```
XPLM_API void XPLMGetFontDimensions(
    XPLMFontID inFontID,
    int * outCharWidth,
    int * outCharHeight,
    int * outDigitsOnly); /* Can be NULL */
```

This routine returns the width and height of a character in a given font. It also tells you if the font only supports numeric digits. Pass NULL if you don't need a given field. Note that for a proportional font the width will be an arbitrary, hopefully average width.

Version 2.0

XPLMMeasureString

```
XPLM_API float XPLMMeasureString(
    XPLMFontID inFontID,
    const char * inChar,
    int inNumChars);
```

This routine returns the width in pixels of a string using a given font. The string is passed as a pointer plus length (and does not need to be null terminated); this is used to allow for measuring substrings. The return value is floating point; it is possible that future font drawing may allow for fractional pixels.

Pages in category "XPLMGraphics"

The following 13 pages are in this category, out of 13 total.

X

- [XPLMBindTexture2d](#)
- [XPLMDrawNumber](#)
- [XPLMDrawString](#)
- [XPLMDrawTranslucentDarkBox](#)
- [XPLMFontID](#)

X cont.

- [XPLMGenerateTextureNumbers](#)
- [XPLMGetFontDimensions](#)
- [XPLMGetTexture](#)
- [XPLMLocalToWorld](#)
- [XPLMMeasureString](#)

X cont.

- [XPLMSetGraphicsState](#)
- [XPLMTextureID](#)
- [XPLMWorldToLocal](#)

X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)
 [Log in](#)

Category:XPLMUtilities

(Redirected from [XPLMUtilities](#))

[Category: Documentation](#)
[Documentation](#)

XPLMUtilities

X-PLANE USER INTERACTION

The user interaction APIs let you simulate commands the user can do with a joystick, keyboard etc. Note that it is generally safer for future compatibility to use one of these commands than to manipulate the underlying sim data.

XPLMCommandKeyID

These enums represent all the keystrokes available within x-plane. They can be sent to x-plane directly. For example, you can reverse thrust using these enumerations.

```
enum {
    xplm_key_pause=0,
    xplm_key_revthrust,
    xplm_key_jettison,
    xplm_key_brakesreg,
    xplm_key_brakesmax,
    xplm_key_gear,
    xplm_key_timedn,
    xplm_key_timeup,
    xplm_key_fadec,
    xplm_key_otto_dis,
    xplm_key_otto_atr,
    xplm_key_otto_asl,
    xplm_key_otto_hdg,
    xplm_key_otto_gps,
    xplm_key_otto_lev,
    xplm_key_otto_hnav,
    xplm_key_otto_alt,
    xplm_key_otto_vvi,
    xplm_key_otto_vnav,
    xplm_key_otto_nav1,
    xplm_key_otto_nav2,
    xplm_key_targ_dn,
    xplm_key_targ_up,
    xplm_key_hdgdn,
    xplm_key_hdgup,
    xplm_key_barodn,
    xplm_key_baroup,
    xplm_key_obs1dn,
    xplm_key_obs1up,
    xplm_key_obs2dn,
    xplm_key_obs2up,
    xplm_key_com1_1,
    xplm_key_com1_2,
    xplm_key_com1_3,
```

```

xplm_key_com1_4,
xplm_key_nav1_1,
xplm_key_nav1_2,
xplm_key_nav1_3,
xplm_key_nav1_4,
xplm_key_com2_1,
xplm_key_com2_2,
xplm_key_com2_3,
xplm_key_com2_4,
xplm_key_nav2_1,
xplm_key_nav2_2,
xplm_key_nav2_3,
xplm_key_nav2_4,
xplm_key_adf_1,
xplm_key_adf_2,
xplm_key_adf_3,
xplm_key_adf_4,
xplm_key_adf_5,
xplm_key_adf_6,
xplm_key_transpon_1,
xplm_key_transpon_2,
xplm_key_transpon_3,
xplm_key_transpon_4,
xplm_key_transpon_5,
xplm_key_transpon_6,
xplm_key_transpon_7,
xplm_key_transpon_8,
xplm_key_flapsup,
xplm_key_flapsdn,
xplm_key_cheatoff,
xplm_key_cheaton,
xplm_key_sbrkoff,
xplm_key_sbrkon,
xplm_key_ailtrimL,
xplm_key_ailtrimR,
xplm_key_rudtrimL,
xplm_key_rudtrimR,
xplm_key_elvtrimD,
xplm_key_elvtrimU,
xplm_key_forward,
xplm_key_down,
xplm_key_left,
xplm_key_right,
xplm_key_back,
xplm_key_tower,
xplm_key_runway,
xplm_key_chase,
xplm_key_free1,
xplm_key_free2,
xplm_key_spot,
xplm_key_fullscrn1,
xplm_key_fullscrn2,
xplm_key_tanspan,
xplm_key_smoke,
xplm_key_map,
xplm_key_zoomin,
xplm_key_zoomout,
xplm_key_cycledump,
xplm_key_replay,
xplm_key_tranID,
xplm_key_max
};
typedef int XPLMCommandKeyID;

```

XPLMCommandButtonID

These are enumerations for all of the things you can do with a joystick button in X-Plane. They currently

match the buttons menu in the equipment setup dialog, but these enums will be stable even if they change in X-Plane.

```
enum {
    xplm_joy_nothing=0,
    xplm_joy_start_all,
    xplm_joy_start_0,
    xplm_joy_start_1,
    xplm_joy_start_2,
    xplm_joy_start_3,
    xplm_joy_start_4,
    xplm_joy_start_5,
    xplm_joy_start_6,
    xplm_joy_start_7,
    xplm_joy_throt_up,
    xplm_joy_throt_dn,
    xplm_joy_prop_up,
    xplm_joy_prop_dn,
    xplm_joy_mixt_up,
    xplm_joy_mixt_dn,
    xplm_joy_carb_tog,
    xplm_joy_carb_on,
    xplm_joy_carb_off,
    xplm_joy_trev,
    xplm_joy_trm_up,
    xplm_joy_trm_dn,
    xplm_joy_rot_trm_up,
    xplm_joy_rot_trm_dn,
    xplm_joy_rud_lft,
    xplm_joy_rud_cntr,
    xplm_joy_rud_rgt,
    xplm_joy_ail_lft,
    xplm_joy_ail_cntr,
    xplm_joy_ail_rgt,
    xplm_joy_B_rud_lft,
    xplm_joy_B_rud_rgt,
    xplm_joy_look_up,
    xplm_joy_look_dn,
    xplm_joy_look_lft,
    xplm_joy_look_rgt,
    xplm_joy_glance_l,
    xplm_joy_glance_r,
    xplm_joy_v_fnh,
    xplm_joy_v_fwh,
    xplm_joy_v_tra,
    xplm_joy_v_twr,
    xplm_joy_v_run,
    xplm_joy_v_cha,
    xplm_joy_v_fr1,
    xplm_joy_v_fr2,
    xplm_joy_v_spo,
    xplm_joy_flapsup,
    xplm_joy_flapsdn,
    xplm_joy_vctswpfwd,
    xplm_joy_vctswpaft,
    xplm_joy_gear_tog,
    xplm_joy_gear_up,
    xplm_joy_gear_down,
    xplm_joy_lft_brake,
    xplm_joy_rgt_brake,
    xplm_joy_brakesREG,
    xplm_joy_brakesMAX,
    xplm_joy_speedbrake,
    xplm_joy_ott_dis,
    xplm_joy_ott_atr,
    xplm_joy_ott_asi,
    xplm_joy_ott_hdg,
    xplm_joy_ott_alt,
```

```

xplm_joy_ott_vvi,
xplm_joy_tim_start,
xplm_joy_tim_reset,
xplm_joy_ecam_up,
xplm_joy_ecam_dn,
xplm_joy_fadec,
xplm_joy_yaw_damp,
xplm_joy_art_stab,
xplm_joy_chute,
xplm_joy_JATO,
xplm_joy_arrest,
xplm_joy_jettison,
xplm_joy_fuel_dump,
xplm_joy_puffsmoke,
xplm_joy_prerot,
xplm_joy_UL_prerot,
xplm_joy_UL_collec,
xplm_joy_TOGA,
xplm_joy_shutdown,
xplm_joy_con_atc,
xplm_joy_fail_now,
xplm_joy_pause,
xplm_joy_rock_up,
xplm_joy_rock_dn,
xplm_joy_rock_lft,
xplm_joy_rock_rgt,
xplm_joy_rock_for,
xplm_joy_rock_aft,
xplm_joy_idle_hilo,
xplm_joy_lanlights,
xplm_joy_max
};
typedef int XPLMCommandButtonID;

```

XPLMHostApplicationID

The plug-in system is based on Austin's cross-platform OpenGL framework and could theoretically be adapted to run in other apps like WorldMaker. The plug-in system also runs against a test harness for internal development and could be adapted to another flight sim (in theory at least). So an ID is providing allowing plug-ins to indentify what app they are running under.

xplm_Host_Unknown	0
xplm_Host_XPlane	1
xplm_Host_PlaneMaker	2
xplm_Host_WorldMaker	3
xplm_Host_Briefer	4
xplm_Host_PartMaker	5
xplm_Host_YoungsMod	6
xplm_Host_XAuto	7

XPLMLanguageCode

These enums define what language the sim is running in. These enumerations do not imply that the sim can or does run in all of these languages; they simply provide a known encoding in the event that a given sim version is localized to a certain language.

xplm_Language_Unknown	0
xplm_Language_English	1

xplm_Language_French	2
xplm_Language_German	3
xplm_Language_Italian	4
xplm_Language_Spanish	5
xplm_Language_Korean	6
xplm_Language_Russian [Version 2.0]	7
xplm_Language_Greek [Version 2.0]	8
xplm_Language_Japanese [Version 2.0]	9
xplm_Language_Chinese [Version 2.0]	10

Version 2.0

XPLMDataFileType

These enums define types of data files you can load or unload using the SDK.

Version 2.0

xplm_DataFile_Situation	1	A situation (.sit) file, which starts off a flight in a given configuration.
xplm_DataFile_ReplayMovie	2	A situation movie (.smo) file, which replays a past flight.

Version 2.0

XPLMError_f

```
typedef void (* XPLMError_f)(  
                                const char *      inMessage);
```

An XPLM error callback is a function that you provide to receive debugging information from the plugin SDK. See XPLMSetErrorCallback for more information. NOTE: for the sake of debugging, your error callback will be called even if your plugin is not enabled, allowing you to receive debug info in your XPluginStart and XPluginStop callbacks. To avoid causing logic errors in the management code, do not call any other plugin routines from your error callback - it is only meant for logging!

XPLMSimulateKeyPress

```
XPLM_API void XPLMSimulateKeyPress(  
                                int      inKeyType,  
                                int      inKey);
```

This function simulates a key being pressed for x-plane. The keystroke goes directly to x-plane; it is never sent to any plug-ins. However, since this is a raw key stroke it may be mapped by the keys file or enter text into a field.

WARNING: This function will be deprecated; do not use it. Instead use XPLMCommandKeyStroke.

XPLMSpeakString

```
XPLM_API void XPLMSpeakString(
    const char * inString);
```

This function displays the string in a translucent overlay over the current display and also speaks the string if text-to-speech is enabled. The string is spoken asynchronously, this function returns immediately.

XPLMCommandKeyStroke

```
XPLM_API void XPLMCommandKeyStroke(
    XPLMCommandKeyID inKey);
```

This routine simulates a command-key stroke. However, the keys are done by function, not by actual letter, so this function works even if the user has remapped their keyboard. Examples of things you might do with this include pausing the simulator.

XPLMCommandButtonPress

```
XPLM_API void XPLMCommandButtonPress(
    XPLMCommandButtonID inButton);
```

This function simulates any of the actions that might be taken by pressing a joystick button. However, this lets you call the command directly rather than have to know which button is mapped where. Important: you must release each button you press. The APIs are separate so that you can 'hold down' a button for a fixed amount of time.

XPLMCommandButtonRelease

```
XPLM_API void XPLMCommandButtonRelease(
    XPLMCommandButtonID inButton);
```

This function simulates any of the actions that might be taken by pressing a joystick button. See XPLMCommandButtonPress

XPLMGetVirtualKeyDescription

```
XPLM_API const char * XPLMGetVirtualKeyDescription(
    char inVirtualKey);
```

Given a virtual key code (as defined in XPLMDefs.h) this routine returns a human-readable string describing the character. This routine is provided for showing users what keyboard mappings they have set up. The string may read 'unknown' or be a blank or NULL string if the virtual key is unknown.

X-PLANE MISC

XPLMReloadScenery

```
XPLM_API void XPLMReloadScenery(void);
```

XPLMReloadScenery reloads the current set of scenery. You can use this function in two typical ways: simply call it to reload the scenery, picking up any new installed scenery, .env files, etc. from disk. Or, change the lat/ref and lon/ref data refs and then call this function to shift the scenery environment.

XPLMGetSystemPath

```
XPLM_API void XPLMGetSystemPath(
    char *
    outSystemPath);
```

This function returns the full path to the X-System folder. Note that this is a directory path, so it ends in a trailing : or /. The buffer you pass should be at least 512 characters long.

XPLMGetPrefsPath

```
XPLM_API void XPLMGetPrefsPath(
    char *
    outPrefsPath);
```

This routine returns a full path to the proper directory to store preferences in. It ends in a : or /. The buffer you pass should be at least 512 characters long.

XPLMGetDirectorySeparator

```
XPLM_API const char * XPLMGetDirectorySeparator(void);
```

This routine returns a string with one char and a null terminator that is the directory separator for the current platform. This allows you to write code that concatenates directory paths without having to #ifdef for platform.

XPLMExtractFileAndPath

```
XPLM_API char * XPLMExtractFileAndPath(
    char *
    inFullPath);
```

Given a full path to a file, this routine separates the path from the file. If the path is a partial directory (e.g. ends in : or \) the trailing directory separator is removed. This routine works in-place; a pointer to the file part of the buffer is returned; the original buffer still starts with the path.

XPLMGetDirectoryContents

```
XPLM_API int XPLMGetDirectoryContents(
    const char *
    inDirectoryPath,
    int
    inFirstReturn,
    char *
    outFileNames,
    int
    inFileNameBufSize,
    char **
    outIndices,
    /* Can be NULL */
    int
    inIndexCount,
    /* Can be NULL */
    int *
    outTotalFiles,
```

```

                                int *
outReturnedFiles);    /* Can be NULL */

```

This routine returns a list of files in a directory (specified by a full path, no trailing : or \). The output is returned as a list of NULL terminated strings. An index array (if specified) is filled with pointers into the strings. This routine The last file is indicated by a zero-length string (and NULL in the indices). This routine will return 1 if you had capacity for all files or 0 if you did not. You can also skip a given number of files.

inDirectoryPath - a null terminated C string containing the full path to the directory with no trailing directory char.

inFirstReturn - the zero-based index of the first file in the directory to return. (Usually zero to fetch all in one pass.)

outFileNames - a buffer to receive a series of sequential null terminated C-string file names. A zero-length C string will be appended to the very end.

inFileNameBufSize - the size of the file name buffer in bytes.

outIndices - a pointer to an array of character pointers that will become an index into the directory. The last file will be followed by a NULL value. Pass NULL if you do not want indexing information.

inIndexCount - the max size of the index in entries.

outTotalFiles - if not NULL, this is filled in with the number of files in the directory.

outReturnedFiles - if not NULL, the number of files returned by this iteration.

Return value - 1 if all info could be returned, 0 if there was a buffer overrun.

WARNING: Before X-Plane 7 this routine did not properly iterate through directories. If X-Plane 6 compatibility is needed, use your own code to iterate directories.

XPLMInitialized

```

XPLM_API int XPLMInitialized(void);

```

This function returns 1 if X-Plane has properly initialized the plug-in system. If this routine returns 0, many XPLM functions will not work.

NOTE: Under normal circumstances a plug-in should never be running while the plug-in manager is not initialized.

WARNING: This function is generally not needed and may be deprecated in the future.

XPLMGetVersions

```
XPLM_API void XPLMGetVersions(
    int *
    outXPlaneVersion,
    int *
    outXPLMVersion,
    XPLMHostApplicationID * outHostID);
```

This routine returns the revision of both X-Plane and the XPLM DLL. All versions are three-digit decimal numbers (e.g. 606 for version 6.06 of X-Plane); the current revision of the XPLM is 200 (2.00). This routine also returns the host ID of the app running us.

The most common use of this routine is to special-case around x-plane version-specific behavior.

XPLMGetLanguage

```
XPLM_API XPLMLanguageCode XPLMGetLanguage(void);
```

This routine returns the language the sim is running in.

XPLMDebugString

```
XPLM_API void XPLMDebugString(
    const char *
    inString);
```

This routine outputs a C-style string to the Log.txt file. The file is immediately flushed so you will not lose data. (This does cause a performance penalty.)

Version 2.0

XPLMSetErrorCallback

```
XPLM_API void XPLMSetErrorCallback(
    XPLMError_f
    inCallback);
```

XPLMSetErrorCallback installs an error-reporting callback for your plugin. Normally the plugin system performs minimum diagnostics to maximize performance. When you install an error callback, you will receive calls due to certain plugin errors, such as passing bad parameters or incorrect data.

The intention is for you to install the error callback during debug sections and put a break-point inside your callback. This will cause you to break into the debugger from within the SDK at the point in your plugin where you made an illegal call.

Installing an error callback may activate error checking code that would not normally run, and this may adversely affect performance, so do not leave error callbacks installed in shipping plugins.

Version 2.0

XPLMFindSymbol

```
XPLM_API void *
XPLMFindSymbol(
    const char *
    inString);
```

This routine will attempt to find the symbol passed in the inString parameter. If the symbol is found a pointer the function is returned, otherwise the function will return NULL.

Version 2.0

XPLMLoadDataFile

```
XPLM_API int
XPLMLoadDataFile(
    XPLMDataFileType
    inFileType,
    const char *
    inFilePath);

/* Can be NULL */
```

Loads a data file of a given type. Paths must be relative to the X-System folder. To clear the replay, pass a NULL file name (this is only valid with replay movies, not sit files).

Version 2.0

XPLMSaveDataFile

```
XPLM_API int
XPLMSaveDataFile(
    XPLMDataFileType
    inFileType,
    const char *
    inFilePath);
```

Saves the current situation or replay; paths are relative to the X-System folder.

Version 2.0

X-PLANE COMMAND MANAGEMENT

The command management APIs let plugins interact with the command-system in X-Plane, the abstraction behind keyboard presses and joystick buttons. This API lets you create new commands and modify the behavior (or get notification) of existing ones.

An X-Plane command consists of three phases: a beginning, continuous repetition, and an ending. The command may be repeated zero times in the event that the user presses a button only momentarily.

XPLMCommandPhase

The phases of a command.

xplm_CommandBegin	0	The command is being started.
xplm_CommandContinue	1	The command is continuing to execute.

xplm_CommandEnd 2 The command has ended.

XPLMCommandRef

```
typedef void * XPLMCommandRef;
```

A command ref is an opaque identifier for an X-Plane command. Command references stay the same for the life of your plugin but not between executions of X-Plane. Command refs are used to execute commands, create commands, and create callbacks for particular commands.

Note that a command is not "owned" by a particular plugin. Since many plugins may participate in a command's execution, the command does not go away if the plugin that created it is unloaded.

XPLMCommandCallback_f

```
typedef int (* XPLMCommandCallback_f)(
    XPLMCommandRef      inCommand,
    XPLMCommandPhase    inPhase,
    void *               inRefcon);
```

A command callback is a function in your plugin that is called when a command is pressed. Your callback receives the command reference for the particular command, the phase of the command that is executing, and a reference pointer that you specify when registering the callback.

Your command handler should return 1 to let processing of the command continue to other plugins and X-Plane, or 0 to halt processing, potentially bypassing X-Plane code.

XPLMFindCommand

```
XPLM_API XPLMCommandRef XPLMFindCommand(
    const char *          inName);
```

XPLMFindCommand looks up a command by name, and returns its command reference or NULL if the command does not exist.

XPLMCommandBegin

```
XPLM_API void XPLMCommandBegin(
    XPLMCommandRef      inCommand);
```

XPLMCommandBegin starts the execution of a command, specified by its command reference. The command is "held down" until XPLMCommandEnd is called.

XPLMCommandEnd

```
XPLM_API void XPLMCommandEnd(
    XPLMCommandRef      inCommand);
```

XPLMCommandEnd ends the execution of a given command that was started with

XPLMCommandBegin.

XPLMCommandOnce

```
XPLM_API void XPLMCommandOnce(
    XPLMCommandRef inCommand);
```

This executes a given command momentarily, that is, the command begins and ends immediately.

XPLMCreateCommand

```
XPLM_API XPLMCommandRef XPLMCreateCommand(
    const char * inName,
    const char * inDescription);
```

XPLMCreateCommand creates a new command for a given string. If the command already exists, the existing command reference is returned. The description may appear in user interface contexts, such as the joystick configuration screen.

XPLMRegisterCommandHandler

```
XPLM_API void XPLMRegisterCommandHandler(
    XPLMCommandRef inComand,
    XPLMCommandCallback_f inHandler,
    int inBefore,
    void * inRefcon);
```

XPLMRegisterCommandHandler registers a callback to be called when a command is executed. You provide a callback with a reference pointer.

If inBefore is true, your command handler callback will be executed before X-Plane executes the command, and returning 0 from your callback will disable X-Plane's processing of the command. If inBefore is false, your callback will run after X-Plane. (You can register a single callback both before and after a command.)

XPLMUnregisterCommandHandler

```
XPLM_API void XPLMUnregisterCommandHandler(
    XPLMCommandRef inComand,
    XPLMCommandCallback_f inHandler,
    int inBefore,
    void * inRefcon);
```

XPLMUnregisterCommandHandler removes a command callback registered with XPLMRegisterCommandHandler.

Pages in category "XPLMUtilities"

The following 36 pages are in this category, out of 36 total.

X

- [XPLMCommandBegin](#)
- [XPLMCommandButtonID](#)
- [XPLMCommandButtonPress](#)
- [XPLMCommandButtonRelease](#)
- [XPLMCommandCallback f](#)
- [XPLMCommandEnd](#)
- [XPLMCommandKeyID](#)
- [XPLMCommandKeyStroke](#)
- [XPLMCommandOnce](#)
- [XPLMCommandPhase](#)
- [XPLMCommandRef](#)
- [XPLMCreateCommand](#)

X cont.

- [XPLMDataFileType](#)
- [XPLMDebugString](#)
- [XPLMError f](#)
- [XPLMExtractFileAndPath](#)
- [XPLMFindCommand](#)
- [XPLMFindSymbol](#)
- [XPLMGetDirectoryContents](#)
- [XPLMGetDirectorySeparator](#)
- [XPLMGetLanguage](#)
- [XPLMGetPrefsPath](#)
- [XPLMGetSystemPath](#)
- [XPLMGetVersions](#)

X cont.

- [XPLMGetVirtualKeyDescription](#)
- [XPLMHostApplicationID](#)
- [XPLMInitialized](#)
- [XPLMLanguageCode](#)
- [XPLMLoadDataFile](#)
- [XPLMRegisterCommandHandler](#)
- [XPLMReloadScenery](#)
- [XPLMSaveDataFile](#)
- [XPLMSetErrorCallback](#)
- [XPLMSimulateKeyPress](#)
- [XPLMSpeakString](#)
- [XPLMUnregisterCommandHandler](#)

X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)

 [Log in](#)

Category:XPLMCamera

(Redirected from [XPLMCamera](#))

[Category: Documentation](#)

[Documentation](#)

XPLMCamera

XPLMCamera - THEORY OF OPERATION The XPLMCamera APIs allow plug-ins to control the camera angle in X-Plane. This has a number of applications, including but not limited to:

- Creating new views (including dynamic/user-controllable views) for the user.
- Creating applications that use X-Plane as a renderer of scenery, aircrafts, or both.

The camera is controlled via six parameters: a location in OpenGL coordinates and pitch, roll and yaw, similar to an airplane's position. OpenGL coordinate info is described in detail in the XPLMGraphics documentation; generally you should use the XPLMGraphics routines to convert from world to local coordinates. The camera's orientation starts facing level with the ground directly up the negative-Z axis (approximately north) with the horizon horizontal. It is then rotated clockwise for yaw, pitched up for positive pitch, and rolled clockwise around the vector it is looking along for roll.

You control the camera either until the user selects a new view or permanently (the later being similar to how UDP camera control works). You control the camera by registering a callback per frame from which you calculate the new camera positions. This guarantees smooth camera motion.

Use the XPLMDataAccess APIs to get information like the position of the aircraft, etc. for complex camera positioning.

CAMERA CONTROL

XPLMCameraControlDuration

This enumeration states how long you want to retain control of the camera. You can retain it indefinitely or until the user selects a new view.

[xplm_ControlCameraUntilViewChanges](#) 1 Control the camera until the user picks a new view.

[xplm_ControlCameraForever](#) 2 Control the camera until your plugin is disabled or another plugin forcably takes control.

XPLMCameraPosition_t

This structure contains a full specification of the camera. X, Y, and Z are the camera's position in OpenGL coordiantes; pitch, roll, and yaw are rotations from a camera facing flat north in degrees. Positive pitch means nose up, positive roll means roll right, and positive yaw means yaw right, all in degrees.

Zoom is a zoom factor, with 1.0 meaning normal zoom and 2.0 magnifying by 2x (objects appear larger).

```
typedef struct {
    float      x;
    float      y;
    float      z;
    float      pitch;
    float      heading;
    float      roll;
    float      zoom;
} XPLMCameraPosition_t;
```

XPLMCameraControl_f

```
typedef int (* XPLMCameraControl_f)(
                                XPLMCameraPosition_t *
outCameraPosition,           /* Can be NULL */
                                int
inIsLosingControl,
                                void *
                                inRefcon);
```

You use an XPLMCameraControl function to provide continuous control over the camera. You are passed in a structure in which to put the new camera position; modify it and return 1 to reposition the camera. Return 0 to surrender control of the camera; camera control will be handled by X-Plane on this draw loop. The contents of the structure as you are called are undefined.

If X-Plane is taking camera control away from you, this function will be called with inIsLosingControl set to 1 and ioCameraPosition NULL.

XPLMControlCamera

```
XPLM_API void XPLMControlCamera(
                                XPLMControlDuration inHowLong,
                                XPLMCameraControl_f inControlFunc,
                                void *
                                inRefcon);
```

This function repositions the camera on the next drawing cycle. You must pass a non-null control function. Specify in inHowLong how long you'd like control (indefinitely or until a key is pressed).

XPLMDontControlCamera

```
XPLM_API void XPLMDontControlCamera(void);
```

This function stops you from controlling the camera. If you have a camera control function, it will not be called with an inIsLosingControl flag. X-Plane will control the camera on the next cycle.

For maximum compatibility you should not use this routine unless you are in possession of the camera.

XPLMIsCameraBeingControlled

```
XPLM_API int XPLMIsCameraBeingControlled(
```

```
                                XPLMCameraControlDuration *
    outCameraControlDuration);    /* Can be NULL */
```

This routine returns 1 if the camera is being controlled, zero if it is not. If it is and you pass in a pointer to a camera control duration, the current control duration will be returned.

XPLMReadCameraPosition

```
XPLM_API void                    XPLMReadCameraPosition(
                                XPLMCameraPosition_t *
    outCameraPosition);
```

This function reads the current camera position.

Pages in category "XPLMCamera"

The following 7 pages are in this category, out of 7 total.

- X
- [XPLMCameraControl f](#)

■ [XPLMCameraControlDuration](#)

■ [XPLMCameraPosition t](#)
- X cont.
- [XPLMControlCamera](#)

■ [XPLMDontControlCamera](#)

■ [XPLMIsCameraBeingControlled](#)
- X cont.
- [XPLMReadCameraPosition](#)

X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)
 [Log in](#)

Category: XPLMPlanes

(Redirected from [XPLMPlanes](#))

[Category: Documentation](#)

[Documentation](#)

XPLMPlanes

The XPLMPlanes APIs allow you to control the various aircraft in x-plane, both the user's and the sim's.

USER AIRCRAFT ACCESS

XPLMSetUsersAircraft

```
XPLM_API void XPLMSetUsersAircraft(
    const char *
    inAircraftPath);
```

This routine changes the user's aircraft. Note that this will reinitialize the user to be on the nearest airport's first runway. Pass in a full path (hard drive and everything including the .acf extension) to the .acf file.

XPLMPlaceUserAtAirport

```
XPLM_API void XPLMPlaceUserAtAirport(
    const char *
    inAirportCode);
```

This routine places the user at a given airport. Specify the airport by its ICAO code (e.g. 'KBOS').

GLOBAL AIRCRAFT ACCESS

XPLM_USER_AIRCRAFT

```
#define XPLM_USER_AIRCRAFT 0
```

The user's aircraft is always index 0.

XPLMPlaneDrawState_t

This structure contains additional plane parameter info to be passed to draw plane. Make sure to fill in the size of the structure field with sizeof(XPLMDrawPlaneState_t) so that the XPLM can tell how many fields you knew about when compiling your plugin (since more fields may be added later).

Most of these fields are ratios from 0 to 1 for control input. X-Plane calculates what the actual controls look like based on the .acf file for that airplane. Note for the yoke inputs, this is what the pilot of the plane

has commanded (post artificial stability system if there were one) and affects ailerons, rudder, etc. It is not necessarily related to the actual position of the plane!

```
typedef struct {
    int                structSize;
```

The size of the draw state struct.

```
float                gearPosition;
```

A ratio from [0..1] describing how far the landing gear is extended.

```
float                flapRatio;
```

Ratio of flap deployment, 0 = up, 1 = full deploy.

```
float                spoilerRatio;
```

Ratio of spoiler deployment, 0 = none, 1 = full deploy.

```
float                speedBrakeRatio;
```

Ratio of speed brake deployment, 0 = none, 1 = full deploy.

```
float                slatRatio;
```

Ratio of slat deployment, 0 = none, 1 = full deploy.

```
float                wingSweep;
```

Wing sweep ratio, 0 = forward, 1 = swept.

```
float                thrust;
```

Thrust power, 0 = none, 1 = full fwd, -1 = full reverse.

```
float                yokePitch;
```

Total pitch input for this plane.

```
float                yokeHeading;
```

Total Heading input for this plane.

```
float                yokeRoll;
```

Total Roll input for this plane.

```
} XPLMPlaneDrawState_t;
```

XPLMCountAircraft

```
XPLM_API void XPLMCountAircraft(
    int *
    outTotalAircraft,
    int *
    outActiveAircraft,
    XPLMPluginID * outController);
```

This function returns the number of aircraft X-Plane is capable of having, as well as the number of aircraft that are currently active. These numbers count the user's aircraft. It can also return the plugin that is currently controlling aircraft. In X-Plane 7, this routine reflects the number of aircraft the user has enabled in the rendering options window.

XPLMGetNthAircraftModel

```
XPLM_API void XPLMGetNthAircraftModel(
    int inIndex,
    char * outFileName,
    char * outPath);
```

This function returns the aircraft model for the Nth aircraft. Indices are zero based, with zero being the user's aircraft. The file name should be at least 256 chars in length; the path should be at least 512 chars in length.

EXCLUSIVE AIRCRAFT ACCESS

The following routines require exclusive access to the airplane APIs. Only one plugin may have this access at a time.

XPLMPlanesAvailable_f

```
typedef void (* XPLMPlanesAvailable_f)(
    void * inRefcon);
```

Your airplanes available callback is called when another plugin gives up access to the multiplayer planes. Use this to wait for access to multiplayer.

XPLMAcquirePlanes

```
XPLM_API int XPLMAcquirePlanes(
    char ** inAircraft,
    /* Can be NULL */
    XPLMPlanesAvailable_f inCallback,
    void * inRefcon);
```

XPLMAcquirePlanes grants your plugin exclusive access to the aircraft. It returns 1 if you gain access, 0 if you do not. inAircraft - pass in an array of pointers to strings specifying the planes you want loaded. For any plane index you do not want loaded, pass a 0-length string. Other strings should be full paths with the .acf extension. NULL terminates this array, or pass NULL if there are no planes you want loaded. If you pass in a callback and do not receive access to the planes your callback will be called when the airplanes are available. If you do receive airplane access, your callback will not be called.

XPLMReleasePlanes

```
XPLM_API void XPLMReleasePlanes(void);
```

Call this function to release access to the planes. Note that if you are disabled, access to planes is released for you and you must reacquire it.

XPLMSetActiveAircraftCount

```
XPLM_API void XPLMSetActiveAircraftCount(
    int inCount);
```

This routine sets the number of active planes. If you pass in a number higher than the total number of planes availables, only the total number of planes available is actually used.

XPLMSetAircraftModel

```
XPLM_API void XPLMSetAircraftModel(
    int inIndex,
    const char * inAircraftPath);
```

This routine loads an aircraft model. It may only be called if you have exclusive access to the airplane APIs. Pass in the path of the model with the .acf extension. The index is zero based, but you may not pass in 0 (use XPLMSetUsersAircraft to load the user's aircraft).

XPLMDisableAIForPlane

```
XPLM_API void XPLMDisableAIForPlane(
    int inPlaneIndex);
```

This routine turns off X-Plane's AI for a given plane. The plane will continue to draw and be a real plane in X-Plane, but will not move itself.

XPLMDrawAircraft

```
XPLM_API void XPLMDrawAircraft(
    int inPlaneIndex,
    float inX,
    float inY,
    float inZ,
    float inPitch,
    float inRoll,
    float inYaw,
    int inFullDraw,
    XPLMPlaneDrawState_t *
    inDrawStateInfo);
```


This routine draws an aircraft. It can only be called from a 3-d drawing callback. Pass in the position of the plane in OpenGL local coordinates and the orientation of the plane. A 1 for full drawing indicates that the whole plane must be drawn; a 0 indicates you only need the nav lights drawn. (This saves rendering time when planes are far away.)

XPLMReinitUsersPlane

```
XPLM_API void XPLMReinitUsersPlane(void);
```

This function recomputes the derived flight model data from the aircraft structure in memory. If you have used the data access layer to modify the aircraft structure, use this routine to resynchronize x-plane; since X-plane works at least partly from derived values, the sim will not behave properly until this is called.

WARNING: this routine does not necessarily place the airplane at the airport; use XPLMSetUsersAircraft to be compatible. This routine is provided to do special experimentation with flight models without resetting flight.

Pages in category "XPLMPanes"

The following 14 pages are in this category, out of 14 total.

X

- [XPLM USER AIRCRAFT](#)
- [XPLMAcquirePlanes](#)
- [XPLMCountAircraft](#)
- [XPLMDisableAIForPlane](#)
- [XPLMDrawAircraft](#)

X cont.

- [XPLMGetNthAircraftModel](#)
- [XPLMPlaceUserAtAiport](#)
- [XPLMPlaneDrawState t](#)
- [XPLMPanesAvailable f](#)
- [XPLMReinitUsersPlane](#)

X cont.

- [XPLMReleasePlanes](#)
- [XPLMSetActiveAircraftCount](#)
- [XPLMSetAircraftModel](#)
- [XPLMSetUsersAircraft](#)



X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)

 [Log in](#)

Category:XPLMNavigation

(Redirected from [XPLMNavigation](#))

[Category: Documentation](#)
[Documentation](#)

XPLMNavigation

XPLMNavigation - THEORY OF OPERATION

The XPLM Navigation APIs give you some access to the navigation databases inside X-Plane. X-Plane stores all navigation information in RAM, so by using these APIs you can gain access to most information without having to go to disk or parse the files yourself.

You can also use this API to program the FMS. You must use the navigation APIs to find the nav-aids you want to program into the FMS, since the FMS is powered internally by x-plane's navigation database.

NAVIGATION DATABASE ACCESS

XPLMNavType

These enumerations define the different types of navaids. They are each defined with a separate bit so that they may be bit-wise added together to form sets of nav-aid types.

NOTE: xplm_Nav_LatLon is a specific lat-lon coordinate entered into the FMS. It will not exist in the database, and cannot be programmed into the FMS. Querying the FMS for navaids will return it. Use XPLMSetFMSEntryLatLon to set a lat/lon waypoint.

xplm_Nav_Unknown	0
xplm_Nav_Airport	1
xplm_Nav_NDB	2
xplm_Nav_VOR	4
xplm_Nav_ILS	8
xplm_Nav_Localizer	16
xplm_Nav_GlideSlope	32
xplm_Nav_OuterMarker	64
xplm_Nav_MiddleMarker	128
xplm_Nav_InnerMarker	256
xplm_Nav_Fix	512
xplm_Nav_DME	1024

XPLMNavRef

```
typedef int XPLMNavRef;
```

XPLMNavRef is an iterator into the navigation database. The navigation database is essentially an array, but it is not necessarily densely populated. The only assumption you can safely make is that like-typed nav-aids are grouped together.

Use XPLMNavRef to refer to a nav-aid.

XPLM_NAV_NOT_FOUND is returned by functions that return an XPLMNavRef when the iterator must be invalid.

XPLM_NAV_NOT_FOUND

```
#define XPLM_NAV_NOT_FOUND -1
```

XPLMGetFirstNavAid

```
XPLM_API XPLMNavRef XPLMGetFirstNavAid(void);
```

This returns the very first navaid in the database. Use this to traverse the entire database. Returns XPLM_NAV_NOT_FOUND if the nav database is empty.

XPLMGetNextNavAid

```
XPLM_API XPLMNavRef XPLMGetNextNavAid(
    XPLMNavRef inNavAidRef);
```

Given a nav aid ref, this routine returns the next navaid. It returns XPLM_NAV_NOT_FOUND if the nav aid passed in was invalid or if the navaid passed in was the last one in the database. Use this routine to iterate across all like-typed navaids or the entire database.

WARNING: due to a bug in the SDK, when fix loading is disabled in the rendering settings screen, calling this routine with the last airport returns a bogus nav aid. Using this nav aid can crash x-plane.

XPLMFindFirstNavAidOfType

```
XPLM_API XPLMNavRef XPLMFindFirstNavAidOfType(
    XPLMNavType inType);
```

This routine returns the ref of the first navaid of the given type in the database or XPLM_NAV_NOT_FOUND if there are no navaids of that type in the database. You must pass exactly one nav aid type to this routine.

WARNING: due to a bug in the SDK, when fix loading is disabled in the rendering settings screen, calling this routine with fixes returns a bogus nav aid. Using this nav aid can crash x-plane.

XPLMFindLastNavAidOfType

```
XPLM_API XPLMNavRef XPLMFindLastNavAidOfType(
    XPLMNavType inType);
```

This routine returns the ref of the last navaid of the given type in the database or XPLM_NAV_NOT_FOUND if there are no navaids of that type in the database. You must pass exactly one nav aid type to this routine.

WARNING: due to a bug in the SDK, when fix loading is disabled in the rendering settings screen, calling this routine with fixes returns a bogus nav aid. Using this nav aid can crash x-plane.

XPLMFindNavAid

```
XPLM_API XPLMNavRef XPLMFindNavAid(
    const char *
inNameFragment, /* Can be NULL */
    const char * inIDFragment,
    /* Can be NULL */
    float * inLat, /*
Can be NULL */
    float * inLon, /*
Can be NULL */
    int * inFrequency,
    /* Can be NULL */
    XPLMNavType inType);
```

This routine provides a number of searching capabilities for the nav database. XPLMFindNavAid will search through every nav aid whose type is within inType (multiple types may be added together) and return any nav-aids found based on the following rules:

If inLat and inLon are not NULL, the navaid nearest to that lat/lon will be returned, otherwise the last navaid found will be returned.

If inFrequency is not NULL, then any navaids considered must match this frequency. Note that this will screen out radio beacons that do not have frequency data published (like inner markers) but not fixes and airports.

If inNameFragment is not NULL, only navaids that contain the fragment in their name will be returned.

If inIDFragment is not NULL, only navaids that contain the fragment in their IDs will be returned.

This routine provides a simple way to do a number of useful searches:

Find the nearest navaid on this frequency. Find the nearest airport. Find the VOR whose ID is "KBOS". Find the nearest airport whose name contains "Chicago".

XPLMGetNavAidInfo

```

XPLM_API void XPLMGetNavAidInfo(
    XPLMNavRef inRef,
    XPLMNavType * outType, /*
    Can be NULL */
    float * outLatitude,
    /* Can be NULL */
    float * outLongitude,
    /* Can be NULL */
    float * outHeight,
    /* Can be NULL */
    int * outFrequency,
    /* Can be NULL */
    float * outHeading,
    /* Can be NULL */
    char * outID, /*
    Can be NULL */
    char * outName, /*
    Can be NULL */
    char * outReg); /*

```

This routine returns information about a navaid. Any non-null field is filled out with information if it is available.

Frequencies are in the nav.dat convention as described in the X-Plane nav database FAQ: NDB frequencies are exact, all others are multiplied by 100.

The buffer for IDs should be at least 6 chars and the buffer for names should be at least 41 chars, but since these values are likely to go up, I recommend passing at least 32 chars for IDs and 256 chars for names when possible.

The outReg parameter tells if the navaid is within the local "region" of loaded DSFs. (This information may not be particularly useful to plugins.) The parameter is a single byte value 1 for true or 0 for false, not a C string.

FLIGHT MANAGEMENT COMPUTER

Note: the FMS works based on an array of entries. Indices into the array are zero-based. Each entry is a nav-aid plus an altitude. The FMS tracks the currently displayed entry and the entry that it is flying to.

The FMS must be programmed with contiguous entries, so clearing an entry at the end shortens the effective flight plan. There is a max of 100 waypoints in the flight plan.

XPLMCountFMSEntries

```

XPLM_API int XPLMCountFMSEntries(void);

```

This routine returns the number of entries in the FMS.

XPLMGetDisplayedFMSEntry

```
XPLM_API int XPLMGetDisplayedFMSEntry(void);
```

This routine returns the index of the entry the pilot is viewing.

XPLMGetDestinationFMSEntry

```
XPLM_API int XPLMGetDestinationFMSEntry(void);
```

This routine returns the index of the entry the FMS is flying to.

XPLMSetDisplayedFMSEntry

```
XPLM_API void XPLMSetDisplayedFMSEntry(int inIndex);
```

This routine changes which entry the FMS is showing to the index specified.

XPLMSetDestinationFMSEntry

```
XPLM_API void XPLMSetDestinationFMSEntry(int inIndex);
```

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

XPLMGetFMSEntryInfo

```
XPLM_API void XPLMGetFMSEntryInfo(
    int inIndex,
    XPLMNavType * outType, /*
    Can be NULL */
    char * outID, /*
    Can be NULL */
    XPLMNavRef * outRef, /*
    Can be NULL */
    int * outAltitude,
    /* Can be NULL */
    float * outLat, /*
    Can be NULL */
    float * outLon); /*
    Can be NULL */
```

This routine returns information about a given FMS entry. A reference to a navaid can be returned allowing you to find additional information (such as a frequency, ILS heading, name, etc.). Some information is available immediately. For a lat/lon entry, the lat/lon is returned by this routine but the navaid cannot be looked up (and the reference will be XPLM_NAV_NOT_FOUND. FMS name entry buffers should be at least 256 chars in length.

XPLMSetFMSEntryInfo

```
XPLM_API void XPLMSetFMSEntryInfo(
    int inIndex,
    XPLMNavRef inRef,
    int inAltitude);
```

This routine changes an entry in the FMS to have the destination navaid passed in and the altitude

specified. Use this only for airports, fixes, and radio-beacon nav aids. Currently of radio beacons, the FMS can only support VORs and NDBs. Use the routines below to clear or fly to a lat/lon.

XPLMSetFMSEntryLatLon

```
XPLM_API void XPLMSetFMSEntryLatLon(
    int inIndex,
    float inLat,
    float inLon,
    int inAltitude);
```

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

XPLMClearFMSEntry

```
XPLM_API void XPLMClearFMSEntry(
    int inIndex);
```

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

GPS RECEIVER

These APIs let you read data from the GPS unit.

XPLMGetGPSDestinationType

```
XPLM_API XPLMNavType XPLMGetGPSDestinationType(void);
```

This routine returns the type of the currently selected GPS destination, one of fix, airport, VOR or NDB.

XPLMGetGPSDestination

```
XPLM_API XPLMNavRef XPLMGetGPSDestination(void);
```

This routine returns the current GPS destination.

Pages in category "XPLMNavigation"

The following 20 pages are in this category, out of 20 total.

X

- [XPLM NAV NOT FOUND](#)
- [XPLMClearFMSEntry](#)
- [XPLMCountFMSEntries](#)
- [XPLMFindFirstNavAidOfType](#)
- [XPLMFindLastNavAidOfType](#)
- [XPLMFindNavAid](#)
- [XPLMGetDestinationFMSEntry](#)

X cont.

- [XPLMGetDisplayedFMSEntry](#)
- [XPLMGetFMSEntryInfo](#)
- [XPLMGetFirstNavAid](#)
- [XPLMGetGPSDestination](#)
- [XPLMGetGPSDestinationType](#)
- [XPLMGetNavAidInfo](#)
- [XPLMGetNextNavAid](#)

X cont.

- [XPLMNavRef](#)
- [XPLMNavType](#)
- [XPLMSetDestinationFMSEntry](#)
- [XPLMSetDisplayedFMSEntry](#)
- [XPLMSetFMSEntryInfo](#)
- [XPLMSetFMSEntryLatLon](#)

Category:XPWidgetDefs

(Redirected from [XPWidgetDefs](#))

[Category: Documentation](#)
[Documentation](#)

XPWidgetDefs

WIDGET DEFINITIONS

A widget is a call-back driven screen entity like a push-button, window, text entry field, etc.

Use the widget API to create widgets of various classes. You can nest them into trees of widgets to create complex user interfaces.

XPWidgetID

```
typedef void * XPWidgetID;
```

A Widget ID is an opaque unique non-zero handle identifying your widget. Use 0 to specify "no widget". This type is defined as wide enough to hold a pointer. You receive a widget ID when you create a new widget and then use that widget ID to further refer to the widget.

XPWidgetPropertyID

Properties are values attached to instances of your widgets. A property is identified by a 32-bit ID and its value is the width of a pointer.

Each widget instance may have a property or not have it. When you set a property on a widget for the first time, the property is added to the widget; it then stays there for the life of the widget.

Some property IDs are predefined by the widget package; you can make up your own property IDs as well.

xpProperty_Refcon	0	A window's refcon is an opaque value used by client code to find other data based on it.
xpProperty_Dragging	1	These properties are used by the utilities to implement dragging.
xpProperty_DragXOff	2	
xpProperty_DragYOff	3	
xpProperty_Hilited	4	Is the widget hilited? (For widgets that support this kind of thing.)
xpProperty_Object	5	Is there a C++ object attached to this widget?
xpProperty_Clip	6	If this property is 1, the widget package will use OpenGL to restrict drawing to the Wiget's exposed rectangle.

`xpProperty_Enabled` 7 Is this widget enabled (for those that have a disabled state too)?

NOTE: Property IDs 1 - 999 are reserved for the widget's library.

`xpProperty_UserStart` 10000 NOTE: Property IDs 1000 - 9999 are allocated to the standard widget classes provided with the library Properties 1000 - 1099 are for widget class 0, 1100 - 1199 for widget class 1, etc.

XPMouseState_t

When the mouse is clicked or dragged, a pointer to this structure is passed to your widget function.

```
typedef struct {  
    int x;  
    int y;  
    int button;
```

Mouse Button number, left = 0 (right button not yet supported).

```
    int delta;
```

Scroll wheel delta (button in this case would be the wheel axis number).

```
} XPMouseState_t;
```

XPKeyState_t

When a key is pressed, a pointer to this struct is passed to your widget function.

```
typedef struct {  
    char key;
```

The ASCII key that was pressed. WARNING: this may be 0 for some non-ASCII key sequences.

```
    XPLMKeyFlags flags;
```

The flags. Make sure to check this if you only want key-downs!

```
    char vkey;
```

The virtual key code for the key

```
} XPKeyState_t;
```

XPWidgetGeometryChange_t

This structure contains the deltas for your widget's geometry when it changes.

```
typedef struct {
    int          dx;
    int          dy;
}
```

+Y = the widget moved up

```
int          dwidth;

int          dheight;

} XPWidgetGeometryChange_t;
```

XPDispatchMode

The dispatching modes describe how the widgets library sends out messages. Currently there are three modes:

<code>xpMode_Direct</code>	0	The message will only be sent to the target widget.
<code>xpMode_UpChain</code>	1	The message is sent to the target widget, then up the chain of parents until the message is handled or a parentless widget is reached.
<code>xpMode_Recursive</code>	2	The message is sent to the target widget and then all of its children recursively depth-first.
<code>xpMode_DirectAllCallbacks</code>	3	The message is sent just to the target, but goes to every callback, even if it is handled.
<code>xpMode_Once</code>	4	The message is only sent to the very first handler even if it is not accepted. (This is really only useful for some internal Widget Lib functions.)

XPWidgetClass

```
typedef int XPWidgetClass;
```

Widget classes define predefined widget types. A widget class basically specifies from a library the widget function to be used for the widget. Most widgets can be made right from classes.

xpWidgetClass_None

```
#define xpWidgetClass_None 0
```

An unspecified widget class. Other widget classes are in `XPStandardWidgets.h`

WIDGET MESSAGES

XPWidgetMessage

Widgets receive 32-bit messages indicating what action is to be taken or notifications of events. The list of messages may be expanded.

<code>xpMsg_None</code>	0	No message, should not be sent.
-------------------------	---	---------------------------------

		<p>The create message is sent once per widget that is created with your widget function and once for any widget that has your widget function attached.</p>
<code>xpMsg_Create</code>	1	<p>Dispatching: Direct</p> <p>Param 1: 1 if you are being added as a subclass, 0 if the widget is first being created.</p>
		<p>The destroy message is sent once for each message that is destroyed that has your widget function.</p>
<code>xpMsg_Destroy</code>	2	<p>Dispatching: Direct for all</p> <p>Param 1: 1 if being deleted by a recursive delete to the parent, 0 for explicit deletion.</p>
		<p>The paint message is sent to your widget to draw itself. The paint message is the bare-bones message; in response you must draw yourself, draw your children, set up clipping and culling, check for visibility, etc. If you don't want to do all of this, ignore the paint message and a draw message (see below) will be sent to you.</p>
<code>xpMsg_Paint</code>	3	<p>Dispatching: Direct</p>
		<p>The draw message is sent to your widget when it is time to draw yourself. OpenGL will be set up to draw in 2-d global screen coordinates, but you should use the XPLM to set up OpenGL state.</p>
<code>xpMsg_Draw</code>	4	<p>Dispatching: Direct</p>
		<p>The key press message is sent once per key that is pressed. The first parameter is the type of key code (integer or char) and the second is the code itself. By handling this event, you consume the key stroke.</p>
<code>xpMsg_KeyPress</code>	5	<p>Handling this message 'consumes' the keystroke; not handling it passes it to your parent widget.</p> <p>Dispatching: Up Chain</p> <p>Param 1: A pointer to an <code>XPKeyState_t</code> structure with the keystroke.</p> <p>Keyboard focus is being given to you. By handling this message you accept keyboard focus. The first parameter will be one if a child of</p>

yours gave up focus to you, 0 if someone set focus on you explicitly.

`xpMsg_KeyTakeFocus` 6 Handling this message accepts focus; not handling refuses focus.
Dispatching: direct

Param 1: 1 if you are gaining focus because your child is giving it up, 0 if someone is explicitly giving you focus.

Keyboard focus is being taken away from you. The first parameter will be one if you are losing focus because another widget is taking it, or 0 if someone called the API to make you lose focus explicitly.

`xpMsg_KeyLoseFocus` 7 Dispatching: Direct

Param 1: 1 if focus is being taken by another widget, 0 if code requested to remove focus.

You receive one mousedown event per click with a mouse-state structure pointed to by parameter 1, by accepting this you eat the click, otherwise your parent gets it. You will not receive drag and mouse up messages if you do not accept the down message.

`xpMsg_MouseDown` 8 Handling this message consumes the mouse click, not handling it passes it to the next widget. You can act 'transparent' as a window by never handling mouse clicks to certain areas.

Dispatching: Up chain NOTE: Technically this is direct dispatched, but the widgets library will shop it to each widget until one consumes the click, making it effectively "up chain".

Param 1: A pointer to an `XPMouseState_t` containing the mouse status.

`xpMsg_MouseDrag` 9 You receive a series of mouse drag messages (typically one per frame in the sim) as the mouse is moved once you have accepted a mouse down message. Parameter one points to a mouse-state structure describing the mouse location. You will continue to receive these until the mouse button is released. You may receive multiple mouse state messages with the same mouse position. You will receive mouse drag events even if the mouse is dragged out of your current or original bounds at the time of the mouse down.

Dispatching: Direct

Param 1: A pointer to an `XPMouseState_t` containing the mouse status.

		<p>The mouseup event is sent once when the mouse button is released after a drag or click. You only receive this message if you accept the mouseDown message. Parameter one points to a mouse state structure.</p>
xpMsg_MouseUp	10	<p>Dispatching: Direct</p>
		<p>Param 1: A pointer to an XPMouseState_t containing the mouse status.</p>
		<p>Your geometry or a child's geometry is being changed.</p>
		<p>Dispatching: Up chain</p>
xpMsg_Reshape	11	<p>Param 1: The widget ID of the original reshaped target.</p>
		<p>Param 2: A pointer to a XPWidgetGeometryChange_t struct describing the change.</p>
		<p>Your exposed area has changed.</p>
xpMsg_ExposedChanged	12	<p>Dispatching: Direct</p>
		<p>A child has been added to you. The child's ID is passed in parameter one.</p>
xpMsg_AcceptChild	13	<p>Dispatching: Direct</p>
		<p>Param 1: The Widget ID of the child being added.</p>
		<p>A child has been removed from to you. The child's ID is passed in parameter one.</p>
xpMsg_LoseChild	14	<p>Dispatching: Direct</p>
		<p>Param 1: The Widget ID of the child being removed.</p>
		<p>You now have a new parent, or have no parent. The parent's ID is passed in, or 0 for no parent.</p>
xpMsg_AcceptParent	15	<p>Dispatching: Direct</p>
		<p>Param 1: The Widget ID of your parent</p>

xpMsg_Shown	16	<p>You or a child has been shown. Note that this does not include you being shown because your parent was shown, you were put in a new parent, your root was shown, etc.</p> <p>Dispatching: Up chain</p> <p>Param 1: The widget ID of the shown widget.</p>
xpMsg_Hidden	17	<p>You have been hidden. See limitations above.</p> <p>Dispatching: Up chain</p> <p>Param 1: The widget ID of the hidden widget.</p>
xpMsg_DescriptorChanged	18	<p>Your descriptor has changed.</p> <p>Dispatching: Direct</p>
xpMsg_PropertyChanged	19	<p>A property has changed. Param 1 contains the property ID.</p> <p>Dispatching: Direct</p> <p>Param 1: The Property ID being changed.</p> <p>Param 2: The new property value</p>
xpMsg_MouseWheel [Version 2.0]	20	<p>The mouse wheel has moved.</p> <p>Return 1 to consume the mouse wheel move, or 0 to pass the message to a parent. Dispatching: Up chain</p> <p>Param 1: A pointer to an XPMouseState_t containing the mouse status.</p>
xpMsg_CursorAdjust [Version 2.0]	21	<p>The cursor is over your widget. If you consume this message, change the XPLMCursorStatus value to indicate the desired result, with the same rules as in XPLMDisplay.h.</p> <p>Return 1 to consume this message, 0 to pass it on.</p> <p>Dispatching: Up chain Param 1: A pointer to an XPMouseState_t struct containing the mouse status.</p>

Param 2: A pointer to a XPLMCursorStatus - set this to the cursor result you desire.

[xpMsg_UserStart](#)

10000

NOTE: Message IDs 1000 - 9999 are allocated to the standard widget classes provided with the library with 1000 - 1099 for widget class 0, 1100 - 1199 for widget class 1, etc. Message IDs 10,000 and beyond are for plugin use.

WIDGET CALLBACK FUNCTION

XPWidgetFunc_t

```
typedef int (* XPWidgetFunc_t)(
    XPWidgetMessage inMessage,
    XPWidgetID      inWidget,
    intptr_t        inParam1,
    intptr_t        inParam2);
```

This function defines your custom widget's behavior. It will be called by the widgets library to send messages to your widget. The message and widget ID are passed in, as well as two ptr-width signed parameters whose meaning varies with the message. Return 1 to indicate that you have processed the message, 0 to indicate that you have not. For any message that is not understood, return 0.

Pages in category "XPWidgetDefs"

The following 10 pages are in this category, out of 10 total.

X

- [XPDispatchMode](#)
- [XPKeyState t](#)
- [XPMouseState t](#)
- [XPWidgetClass](#)

X cont.

- [XPWidgetFunc t](#)
- [XPWidgetGeometryChange t](#)
- [XPWidgetID](#)
- [XPWidgetMessage](#)

X cont.

- [XPWidgetPropertyID](#)
- [XpWidgetClass None](#)

X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)
 [Log in](#)

Category:XPWidgets

(Redirected from [XPWidgets](#))

[Category: Documentation](#)

[Documentation](#)

XPWidgets

WIDGETS - THEORY OF OPERATION AND NOTES

Widgets are persistent view 'objects' for X-Plane. A widget is an object referenced by its opaque handle (widget ID) and the APIs in this file. You cannot access the widget's guts directly. Every Widget has the following intrinsic data:

- A bounding box defined in global screen coordinates with 0,0 in the bottom left and +y = up, +x = right.
- A visible box, which is the intersection of the bounding box with the widget's parents visible box.
- Zero or one parent widgets. (Always zero if the widget is a root widget.
- Zero or more child widgets.
- Whether the widget is a root. Root widgets are the top level plugin windows.
- Whether the widget is visible.
- A text string descriptor, whose meaning varies from widget to widget.
- An arbitrary set of 32 bit integral properties defined by 32-bit integral keys. This is how specific widgets store specific data.
- A list of widget callbacks proc that implements the widgets behaviors.

The Widgets library sends messages to widgets to request specific behaviors or notify the widget of things.

Widgets may have more than one callback function, in which case messages are sent to the most recently added callback function until the message is handled. Messages may also be sent to parents or children; see the XPWidgetDefs.h header file for the different widget message dispatching functions. By adding a callback function to a window you can 'subclass' its behavior.

A set of standard widgets are provided that serve common UI purposes. You can also customize or implement entirely custom widgets.

Widgets are different than other view hierarchies (most notably Win32, which they bear a striking resemblance to) in the following ways:

- Not all behavior can be patched. State that is managed by the XPWidgets DLL and not by individual widgets cannot be customized.
- All coordinates are in global screen coordinates. Coordinates are not relative to an enclosing widget, nor are they relative to a display window.
- Widget messages are always dispatched synchronously, and there is no concept of scheduling an update or a dirty region. Messages originate from X-Plane as the sim cycle goes by. Since x-plane is constantly redrawing, so are widgets; there is no need to mark a part of a widget as 'needing redrawing' because redrawing happens frequently whether the widget needs it or not.
- Any widget may be a 'root' widget, causing it to be drawn; there is no relationship between widget class and rootness. Root widgets are implemented as XPLMDisplay windows.

WIDGET CREATION AND MANAGEMENT

XPCreateWidget

```
WIDGET_API XPWidgetID      XPCreateWidget(
    int                    inLeft,
    int                    inTop,
    int                    inRight,
    int                    inBottom,
    int                    inVisible,
    const char *           inDescriptor,
    int                    inIsRoot,
    XPWidgetID             inContainer,
    XPWidgetClass           inClass);
```

This function creates a new widget and returns the new widget's ID to you. If the widget creation fails for some reason, it returns NULL. Widget creation will fail either if you pass a bad class ID or if there is not adequate memory.

Input Parameters:

- Top, left, bottom, and right in global screen coordinates defining the widget's location on the screen.
- inVisible is 1 if the widget should be drawn, 0 to start the widget as hidden.
- inDescriptor is a null terminated string that will become the widget's descriptor.
- inIsRoot is 1 if this is going to be a root widget, 0 if it will not be.
- inContainer is the ID of this widget's container. It must be 0 for a root widget. for a non-root widget, pass the widget ID of the widget to place this widget within. If this widget is not going to start inside another widget, pass 0; this new widget will then just be floating off in space (and will not be drawn until it is placed in a widget).

- inClass is the class of the widget to draw. Use one of the predefined class-IDs to create a standard widget.

A note on widget embedding: a widget is only called (and will be drawn, etc.) if it is placed within a widget that will be called. Root widgets are always called. So it is possible to have whole chains of widgets that are simply not called. You can preconstruct widget trees and then place them into root widgets later to activate them if you wish.

XPCreateCustomWidget

```
WIDGET_API XPWidgetID      XPCreateCustomWidget(
    int                    inLeft,
    int                    inTop,
    int                    inRight,
    int                    inBottom,
    int                    inVisible,
    const char *           inDescriptor,
    int                    inIsRoot,
    XPWidgetID             inContainer,
    XPWidgetFunc_t         inCallback);
```

This function is the same as XPCreateWidget except that instead of passing a class ID, you pass your widget callback function pointer defining the widget. Use this function to define a custom widget. All parameters are the same as XPCreateWidget, except that the widget class has been replaced with the widget function.

XPDestroyWidget

```
WIDGET_API void            XPDestroyWidget(
    XPWidgetID             inWidget,
    int                    inDestroyChildren);
```

This class destroys a widget. Pass in the ID of the widget to kill. If you pass 1 for inDestroyChildren, the widget's children will be destroyed first, then this widget will be destroyed. (Furthermore, the widget's children will be destroyed with the inDestroyChildren flag set to 1, so the destruction will recurse down the widget tree.) If you pass 0 for this flag, the child widgets will simply end up with their parent set to 0.

XPSendMessageToWidget

```
WIDGET_API int             XPSendMessageToWidget(
    XPWidgetID             inWidget,
    XPWidgetMessage        inMessage,
    XPDispatchMode         inMode,
    intptr_t               inParam1,
    intptr_t               inParam2);
```

This sends any message to a widget. You should probably not go around simulating the predefined messages that the widgets library defines for you. You may however define custom messages for your widgets and send them with this method.

This method supports several dispatching patterns; see XPDispatchMode for more info. The function returns 1 if the message was handled, 0 if it was not.

For each widget that receives the message (see the dispatching modes), each widget function from the most recently installed to the oldest one receives the message in order until it is handled.

WIDGET POSITIONING AND VISIBILITY

XPPlaceWidgetWithin

```
WIDGET_API void                XPPlaceWidgetWithin(
                                XPWidgetID          inSubWidget,
                                XPWidgetID          inContainer);
```

This function changes which container a widget resides in. You may NOT use this function on a root widget! inSubWidget is the widget that will be moved. Pass a widget ID in inContainer to make inSubWidget be a child of inContainer. It will become the last/closest widget in the container. Pass 0 to remove the widget from any container. Any call to this other than passing the widget ID of the old parent of the affected widget will cause the widget to be removed from its old parent. Placing a widget within its own parent simply makes it the last widget.

NOTE: this routine does not reposition the sub widget in global coordinates. If the container has layout management code, it will reposition the subwidget for you, otherwise you must do it with SetWidgetGeometry.

XPCountChildWidgets

```
WIDGET_API int                XPCountChildWidgets(
                                XPWidgetID          inWidget);
```

This routine returns the number of widgets another widget contains.

XPGetNthChildWidget

```
WIDGET_API XPWidgetID         XPGetNthChildWidget(
                                XPWidgetID          inWidget,
                                int                  inIndex);
```

This routine returns the widget ID of a child widget by index. Indexes are 0 based, from 0 to one minus the number of widgets in the parent, inclusive. If the index is invalid, 0 is returned.

XPGetParentWidget

```
WIDGET_API XPWidgetID         XPGetParentWidget(
                                XPWidgetID          inWidget);
```

This routine returns the parent of a widget, or 0 if the widget has no parent. Root widgets never have parents and therefore always return 0.

XPShowWidget

```
WIDGET_API void XPShowWidget(
    XPWidgetID inWidget);
```

This routine makes a widget visible if it is not already. Note that if a widget is not in a rooted widget hierarchy or one of its parents is not visible, it will still not be visible to the user.

XPHideWidget

```
WIDGET_API void XPHideWidget(
    XPWidgetID inWidget);
```

Makes a widget invisible. See XPShowWidget for considerations of when a widget might not be visible despite its own visibility state.

XPWidgetVisible

```
WIDGET_API int XPWidgetVisible(
    XPWidgetID inWidget);
```

This returns 1 if a widget is visible, 0 if it is not. Note that this routine takes into consideration whether a parent is invisible. Use this routine to tell if the user can see the widget.

XPFindRootWidget

```
WIDGET_API XPWidgetID XPFindRootWidget(
    XPWidgetID inWidget);
```

XPFindRootWidget returns the Widget ID of the root widget that contains the passed in widget or NULL if the passed in widget is not in a rooted hierarchy.

XPBringRootWidgetToFront

```
WIDGET_API void XPBringRootWidgetToFront(
    XPWidgetID inWidget);
```

This routine makes the specified widget be in the front most widget hierarchy. If this widget is a root widget, its widget hierarchy comes to front, otherwise the widget's root is brought to the front. If this widget is not in an active widget hierarchy (e.g. there is no root widget at the top of the tree), this routine does nothing.

XPWidgetInFront

```
WIDGET_API int XPWidgetInFront(
    XPWidgetID inWidget);
```

This routine returns true if this widget's hierarchy is the front most hierarchy. It returns false if the widget's hierarchy is not in front, or if the widget is not in a rooted hierarchy.

XPGetWidgetGeometry

```
WIDGET_API void XPGetWidgetGeometry(
    XPWidgetID inWidget,
```

```

Can be NULL */
Can be NULL */
/* Can be NULL */
/* Can be NULL */

int * outLeft, /*
int * outTop, /*
int * outRight,
int * outBottom);

```

This routine returns the bounding box of a widget in global coordinates. Pass NULL for any parameter you are not interested in.

XPSetWidgetGeometry

```

WIDGET_API void XPSetWidgetGeometry(
XPWidgetID inWidget,
int inLeft,
int inTop,
int inRight,
int inBottom);

```

This function changes the bounding box of a widget.

XPGetWidgetForLocation

```

WIDGET_API XPWidgetID XPGetWidgetForLocation(
XPWidgetID inContainer,
int inXOffset,
int inYOffset,
int inRecursive,
int inVisibleOnly);

```

Given a widget and a location, this routine returns the widget ID of the child of that widget that owns that location. If inRecursive is true then this will return a child of a child of a widget as it tries to find the deepest widget at that location. If inVisibleOnly is true, then only visible widgets are considered, otherwise all widgets are considered. The widget ID passed for inContainer will be returned if the location is in that widget but not in a child widget. 0 is returned if the location is not in the container.

NOTE: if a widget's geometry extends outside its parents geometry, it will not be returned by this call for mouse locations outside the parent geometry. The parent geometry limits the child's eligibility for mouse location.

XPGetWidgetExposedGeometry

```

WIDGET_API void XPGetWidgetExposedGeometry(
XPWidgetID inWidgetID,
int * outLeft, /*
Can be NULL */
int * outTop, /*
Can be NULL */
int * outRight,
/* Can be NULL */
int * outBottom);
/* Can be NULL */

```

This routine returns the bounds of the area of a widget that is completely within its parent widgets. Since a widget's bounding box can be outside its parent, part of its area will not be eligible for mouse clicks and should not draw. Use `XPGetWidgetGeometry` to find out what area defines your widget's shape, but use this routine to find out what area to actually draw into. Note that the widget library does not use OpenGL clipping to keep frame rates up, although you could use it internally.

ACCESSING WIDGET DATA

XPSetWidgetDescriptor

```
WIDGET_API void                XPSetWidgetDescriptor(
                                XPWidgetID           inWidget,
                                const char *          inDescriptor);
```

Every widget has a descriptor, which is a text string. What the text string is used for varies from widget to widget; for example, a push button's text is its descriptor, a caption shows its descriptor, and a text field's descriptor is the text being edited. In other words, the usage for the text varies from widget to widget, but this API provides a universal and convenient way to get at it. While not all UI widgets need their descriptor, many do.

XPGetWidgetDescriptor

```
WIDGET_API int                XPGetWidgetDescriptor(
                                XPWidgetID           inWidget,
                                char *                outDescriptor,
                                int                   inMaxDescLength);
```

This routine returns the widget's descriptor. Pass in the length of the buffer you are going to receive the descriptor in. The descriptor will be null terminated for you. This routine returns the length of the actual descriptor; if you pass NULL for outDescriptor, you can get the descriptor's length without getting its text. If the length of the descriptor exceeds your buffer length, the buffer will not be null terminated (this routine has 'strncpy' semantics).

XPSetWidgetProperty

```
WIDGET_API void                XPSetWidgetProperty(
                                XPWidgetID           inWidget,
                                XPWidgetPropertyID    inProperty,
                                intptr_t              inValue);
```

This function sets a widget's property. Properties are arbitrary values associated by a widget by ID.

XPGetWidgetProperty

```
WIDGET_API intptr_t            XPGetWidgetProperty(
                                XPWidgetID           inWidget,
                                XPWidgetPropertyID    inProperty,
                                int *                 inExists);

/* Can be NULL */
```

This routine returns the value of a widget's property, or 0 if the property is not defined. If you need to know whether the property is defined, pass a pointer to an int for inExists; the existence of that property

will be returned in the int. Pass NULL for inExists if you do not need this information.

KEYBOARD MANAGEMENT

XPSetKeyboardFocus

```
WIDGET_API XPWidgetID      XPSetKeyboardFocus(
                                XPWidgetID      inWidget);
```

XPSetKeyboardFocus controls which widget will receive keystrokes. Pass the Widget ID of the widget to get the keys. Note that if the widget does not care about keystrokes, they will go to the parent widget, and if no widget cares about them, they go to X-Plane.

If you set the keyboard focus to Widget ID 0, X-Plane gets keyboard focus.

This routine returns the widget ID that ended up with keyboard focus, or 0 for x-plane.

Keyboard focus is not changed if the new widget will not accept it. For setting to x-plane, keyboard focus is always accepted.

XP LoseKeyboardFocus

```
WIDGET_API void      XP LoseKeyboardFocus(
                                XPWidgetID      inWidget);
```

This causes the specified widget to lose focus; focus is passed to its parent, or the next parent that will accept it. This routine does nothing if this widget does not have focus.

XPGetWidgetWithFocus

```
WIDGET_API XPWidgetID      XPGetWidgetWithFocus(void);
```

This routine returns the widget that has keyboard focus, or 0 if X-Plane has keyboard focus or some other plugin window that does not have widgets has focus.

CREATING CUSTOM WIDGETS

XPAddWidgetCallback

```
WIDGET_API void      XPAddWidgetCallback(
                                XPWidgetID      inWidget,
                                XPWidgetFunc_t    inNewCallback);
```

This function adds a new widget callback to a widget. This widget callback supercedes any existing ones and will receive messages first; if it does not handle messages they will go on to be handled by pre-existing widgets.

The widget function will remain on the widget for the life of the widget. The creation message will be sent to the new callback immediately with the widget ID, and the destruction message will be sent before the

other widget function receives a destruction message.

This provides a way to 'subclass' an existing widget. By providing a second hook that only handles certain widget messages, you can customize or extend widget behavior.

XPGetWidgetClassFunc

```
WIDGET_API XPWidgetFunc_t      XPGetWidgetClassFunc(
                                XPWidgetClass      inWidgetClass);
```

Given a widget class, this function returns the callbacks that power that widget class.

Pages in category "XPWidgets"

The following 27 pages are in this category, out of 27 total.

- X
- [XPAddWidgetCallback](#)
 - [XPBringRootWidgetToFront](#)
 - [XPCountChildWidgets](#)
 - [XPCreateCustomWidget](#)
 - [XPCreateWidget](#)
 - [XPDestroyWidget](#)
 - [XPFindRootWidget](#)
 - [XPGetNthChildWidget](#)
 - [XPGetParentWidget](#)
- X cont.
- [XPGetWidgetClassFunc](#)
 - [XPGetWidgetDescriptor](#)
 - [XPGetWidgetExposedGeometry](#)
 - [XPGetWidgetForLocation](#)
 - [XPGetWidgetGeometry](#)
 - [XPGetWidgetProperty](#)
 - [XPGetWidgetWithFocus](#)
 - [XPHideWidget](#)
 - [XPIsWidgetInFront](#)
- X cont.
- [XPIsWidgetVisible](#)
 - [XP LoseKeyboardFocus](#)
 - [XPPlaceWidgetWithin](#)
 - [XPSendMessageToWidget](#)
 - [XPSetKeyboardFocus](#)
 - [XPSetWidgetDescriptor](#)
 - [XPSetWidgetGeometry](#)
 - [XPSetWidgetProperty](#)
 - [XPShowWidget](#)

X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)
 [Log in](#)

Category:XPWidgetUtils

(Redirected from [XPWidgetUtils](#))

[Category: Documentation](#)
[Documentation](#)

XPWidgetUtils

XPWidgetUtils - USAGE NOTES

The XPWidgetUtils library contains useful functions that make writing and using widgets less of a pain.

One set of functions are the widget behavior functions. These functions each add specific useful behaviors to widgets. They can be used in two manners:

1. You can add a widget behavior function to a widget as a callback proc using the `XPAddWidgetCallback` function. The widget will gain that behavior. Remember that the last function you add has highest priority. You can use this to change or augment the behavior of an existing finished widget.
2. You can call a widget function from inside your own widget function. This allows you to include useful behaviors in custom-built widgets. A number of the standard widgets get their behavior from this library. To do this, call the behavior function from your function first. If it returns 1, that means it handled the event and you don't need to; simply return 1.

GENERAL UTILITIES

XPWidgetCreate_t

This structure contains all of the parameters needed to create a widget. It is used with `XPUCreateWidgets` to create widgets in bulk from an array. All parameters correspond to those of `XPCreateWidget` except for the container index. If the container index is equal to the index of a widget in the array, the widget in the array passed to `XPUCreateWidgets` is used as the parent of this widget. Note that if you pass an index greater than your own position in the array, the parent you are requesting will not exist yet. If the container index is `NO_PARENT`, the parent widget is specified as `NULL`. If the container index is `PARAM_PARENT`, the widget passed into `XPUCreateWidgets` is used.

```
typedef struct {
    int             left;
    int             top;
    int             right;
    int             bottom;
    int             visible;
    const char *    descriptor;
    int             isRoot;
```

```
int
XPWidgetClass
} XPWidgetCreate_t;
```

```
containerIndex;
widgetClass;
```

NO_PARENT

```
#define NO_PARENT -1
```

PARAM_PARENT

```
#define PARAM_PARENT -2
```

XPUCreateWidgets

```
WIDGET_API void
inWidgetDefs,

XPUCreateWidgets(
    const XPWidgetCreate_t *
    int
    XPWidgetID
    XPWidgetID *
    inCount,
    inParamParent,
    ioWidgets);
```

This function creates a series of widgets from a table...see XPCreateWidget_t above. Pass in an array of widget creation structures and an array of widget IDs that will receive each widget.

Widget parents are specified by index into the created widget table, allowing you to create nested widget structures. You can create multiple widget trees in one table. Generally you should create widget trees from the top down.

You can also pass in a widget ID that will be used when the widget's parent is listed as PARAM_PARENT; this allows you to embed widgets created with XPUCreateWidgets in a widget created previously.

XPUMoveWidgetBy

```
WIDGET_API void
XPUMoveWidgetBy(
    XPWidgetID
    int
    int
    inWidget,
    inDeltaX,
    inDeltaY);
```

Simply moves a widget by an amount, +x = right, +y=up, without resizing the widget.

LAYOUT MANAGERS

The layout managers are widget behavior functions for handling where widgets move. Layout managers can be called from a widget function or attached to a widget later.

XPUFixedLayout

```
WIDGET_API int
XPWidgetFixedLayout(
    XPWidgetMessage inMessage,
    XPWidgetID inWidget,
    intptr_t inParam1,
    intptr_t inParam2);
```

This function causes the widget to maintain its children in fixed position relative to itself as it is resized. Use this on the top level 'window' widget for your window.

WIDGET PROC BEHAVIORS

These widget behavior functions add other useful behaviors to widgets. These functions cannot be attached to a widget; they must be called from your widget function.

XPWidgetSelectIfNeeded

```
WIDGET_API int
XPWidgetSelectIfNeeded(
    XPWidgetMessage inMessage,
    XPWidgetID inWidget,
    intptr_t inParam1,
    intptr_t inParam2,
    int inEatClick);
```

This causes the widget to bring its window to the foreground if it is not already. inEatClick specifies whether clicks in the background should be consumed by bringing the window to the foreground.

XPWidgetDefocusKeyboard

```
WIDGET_API int
XPWidgetDefocusKeyboard(
    XPWidgetMessage inMessage,
    XPWidgetID inWidget,
    intptr_t inParam1,
    intptr_t inParam2,
    int inEatClick);
```

This causes a click in the widget to send keyboard focus back to X-Plane. This stops editing of any text fields, etc.

XPWidgetDragWidget

```
WIDGET_API int
XPWidgetDragWidget(
    XPWidgetMessage inMessage,
    XPWidgetID inWidget,
    intptr_t inParam1,
    intptr_t inParam2,
    int inLeft,
    int inTop,
    int inRight,
    int inBottom);
```

XPWidgetDragWidget drags the widget in response to mouse clicks. Pass in not only the event, but the global coordinates of the drag region, which might be a sub-region of your widget (for example, a title bar).

The following 9 pages are in this category, out of 9 total.

N

- [NO PARENT](#)

P

- [PARAM PARENT](#)

X

- [XPUCreateWidgets](#)

X cont.

- [XPUDefocusKeyboard](#)
- [XPUDragWidget](#)
- [XPUFixedLayout](#)

X cont.

- [XPUMoveWidgetBy](#)
- [XPUSelectIfNeeded](#)
- [XPWidgetCreate t](#)

Category:XPStandardWidgets

(Redirected from [XPStandardWidgets](#))

[Category: Documentation](#)
[Documentation](#)

XPStandardWidgets

XPStandardWidgets - THEORY OF OPERATION

The standard widgets are widgets built into the widgets library. While you can gain access to the widget function that drives them, you generally use them by calling `XPCreateWidget` and then listening for special messages, etc.

The standard widgets often send messages to themselves when the user performs an event; these messages are sent up the widget hierarchy until they are handled. So you can add a widget proc directly to a push button (for example) to intercept the message when it is clicked, or you can put one widget proc on a window for all of the push buttons in the window. Most of these messages contain the original widget ID as a parameter so you can know which widget is messaging no matter who it is sent to.

MAIN WINDOW

The main window widget class provides a "window" as the user knows it. These windows are draggable and can be selected. Use them to create floating windows and non-modal dialogs.

xpWidgetClass_MainWindow

```
#define xpWidgetClass_MainWindow 1
```

Main Window Type Values

These type values are used to control the appearance of a main window.

- `xpMainWindowStyle_MainWindow` 0
- The standard main window; pin stripes on XP7, metal frame on XP6.
- `xpMainWindowStyle_Translucent` 1
- A translucent dark gray window, like the one ATC messages appear in.

Main Window Properties

- `xpProperty_MainWindowType` 1100
- This property specifies the type of window. Set to one of the main window types above.
This property specifies whether the main window has

`xpProperty_MainWindowHasCloseBoxes` 1200 close boxes in its corners.

MainWindow Messages

`xpMessage_CloseButtonPushed` 1200 This message is sent when the close buttons are pressed for your window.

SUB WINDOW

X-plane dialogs are divided into separate areas; the sub window widgets allow you to make these areas. Create one main window and place several subwindows inside it. Then place your controls inside the subwindows.

xpWidgetClass_SubWindow

```
#define xpWidgetClass_SubWindow 2
```

SubWindow Type Values

These values control the appearance of the subwindow.

`xpSubWindowStyle_SubWindow` 0 A panel that sits inside a main window.

`xpSubWindowStyle_Screen` 2 A screen that sits inside a panel for showing text information.

`xpSubWindowStyle_ListView` 3 A list view for scrolling lists.

SubWindow Properties

`xpProperty_SubWindowType` 1200 This property specifies the type of window. Set to one of the subwindow types above.

BUTTON

The button class provides a number of different button styles and behaviors, including push buttons, radio buttons, check boxes, etc. The button label appears on or next to the button depending on the button's appearance, or type.

The button's behavior is a separate property that dictates who it highlights and what kinds of messages it sends. Since behavior and type are different, you can do strange things like make check boxes that act as push buttons or push buttons with radio button behavior.

In X-Plane 6 there were no check box graphics. The result is the following behavior: in x-plane 6 all check box and radio buttons are round (radio-button style) buttons; in X-Plane 7 they are all square (check-box style) buttons. In a future version of x-plane, the `xpButtonBehavior` enums will provide the correct graphic (check box or radio button) giving the expected result.

xpWidgetClass_Button

```
#define xpWidgetClass_Button 3
```

Button Types

These define the visual appearance of buttons but not how they respond to the mouse.

<code>xpPushButton</code>	0	This is a standard push button, like an "OK" or "Cancel" button in a dialog box.
<code>xpRadioButton</code>	1	A check box or radio button. Use this and the button behaviors below to get the desired behavior.
<code>xpWindowCloseBox</code>	3	A window close box.
<code>xpLittleDownArrow</code>	5	A small down arrow.
<code>xpLittleUpArrow</code>	6	A small up arrow.

Button Behavior Values

These define how the button responds to mouse clicks.

<code>xpButtonBehaviorPushButton</code>	0	Standard push button behavior. The button hilites while the mouse is clicked over it and unhilites when the mouse is moved outside of it or released. If the mouse is released over the button, the <code>xpMsg_PushButtonPressed</code> message is sent.
<code>xpButtonBehaviorCheckBox</code>	1	Check box behavior. The button immediately toggles its value when the mouse is clicked and sends out a <code>xpMsg_ButtonStateChanged</code> message.
<code>xpButtonBehaviorRadioButton</code>	2	Radio button behavior. The button immediately sets its state to one and sends out a <code>xpMsg_ButtonStateChanged</code> message if it was not already set to one. You must turn off other radio buttons in a group in your code.

Button Properties

<code>xpProperty_ButtonType</code>	1300	This property sets the visual type of button. Use one of the button types above.
<code>xpProperty_ButtonBehavior</code>	1301	This property sets the button's behavior. Use one of the button behaviors above.
<code>xpProperty_ButtonState</code>	1302	This property tells whether a check box or radio button is "checked" or not. Not used for push buttons.

Button Messages

These messages are sent by the button to itself and then up the widget chain when the button is clicked. (You may intercept them by providing a widget handler for the button itself or by providing a handler in a parent widget.)

<code>xpMsg_PushButtonPressed</code>	1300	This message is sent when the user completes a click and release in a button with push button behavior. Parameter one of the message is the widget ID of the button. This message is dispatched up the widget hierarchy.
<code>xpMsg_ButtonStateChanged</code>	1301	This message is sent when a button is clicked that has radio button or check box behavior and its value changes. (Note that if the value changes by setting a property you do not receive this message!) Parameter one is the widget ID of the button, parameter 2 is the new

state value, either zero or one. This message is dispatched up the widget hierarchy.

TEXT FIELD

The text field widget provides an editable text field including mouse selection and keyboard navigation. The contents of the text field are its descriptor. (The descriptor changes as the user types.)

The text field can have a number of types, that effect the visual layout of the text field. The text field sends messages to itself so you may control its behavior.

If you need to filter keystrokes, add a new handler and intercept the key press message. Since key presses are passed by pointer, you can modify the keystroke and pass it through to the text field widget.

WARNING: in x-plane before 7.10 (including 6.70) null characters could crash x-plane. To prevent this, wrap this object with a filter function (more instructions can be found on the SDK website).

xpWidgetClass_TextField

```
#define xpWidgetClass_TextField 4
```

Text Field Type Values

These control the look of the text field.

- `xpTextEntryField` 0 A field for text entry.
A transparent text field. The user can type and the text is drawn, but no background is drawn. You can draw your own background by adding a widget handler and prehandling the draw message.
- `xpTextTransparent` 3 A translucent edit field, dark gray.

Text Field Properties

		This is the character position the selection starts at, zero based.
<code>xpProperty_EditFieldSelStart</code>	1400	If it is the same as the end insertion point, the insertion point is not a selection.
<code>xpProperty_EditFieldSelEnd</code>	1401	This is the character position of the end of the selection.
<code>xpProperty_EditFieldSelDragStart</code>	1402	This is the character position a drag was started at if the user is dragging to select text, or -1 if a drag is not in progress.
<code>xpProperty_TextFieldType</code>	1403	This is the type of text field to display, from the above list.
<code>xpProperty_PasswordMode</code>	1404	Set this property to 1 to password protect the field. Characters will be drawn as *s even though the descriptor will contain plain-text.
<code>xpProperty_MaxCharacters</code>	1405	The max number of characters you can enter, if limited. Zero means unlimited.
<code>xpProperty_ScrollPosition</code>	1406	The first visible character on the left. This effectively scrolls the text field.
<code>xpProperty_Font</code>	1407	The font to draw the field's text with. (An XPLMFontID.)

xpProperty_ActiveEditSide

1408 This is the active side of the insert selection. (Internal)

Text Field Messages

Text Field Messages

xpMsg_TextFieldChanged

1400 The text field sends this message to itself when its text changes. It sends the message up the call chain; param1 is the text field's widget ID.

SCROLL BAR

A standard scroll bar or slider control. The scroll bar has a minimum, maximum and current value that is updated when the user drags it. The scroll bar sends continuous messages as it is dragged.

xpWidgetClass_ScrollBar

```
#define xpWidgetClass_ScrollBar 5
```

Scroll Bar Type Values

This defines how the scroll bar looks.

Scroll bar types.

xpScrollBarTypeScrollBar

0

A standard x-plane scroll bar (with arrows on the ends).

xpScrollBarTypeSlider

1

A slider, no arrows.

Scroll Bar Properties

- xpProperty_ScrollBarSliderPosition
- 1500
- The current position of the thumb (in between the min and max, inclusive)
- xpProperty_ScrollBarMin
- 1501
- The value the scroll bar has when the thumb is in the lowest position.
- xpProperty_ScrollBarMax
- 1502
- The value the scroll bar has when the thumb is in the highest position.
- xpProperty_ScrollBarPageAmount
- 1503
- How many units to move the scroll bar when clicking next to the thumb. The scroll bar always moves one unit when the arrows are clicked.
- xpProperty_ScrollBarType
- 1504
- The type of scrollbar from the enums above.
- xpProperty_ScrollBarSlop
- 1505
- Used internally.

Scroll Bar Messages

The Scroll Bar sends this message when the slider

`xpMsg_ScrollBarSliderPositionChanged` 1500 position changes. It sends the message up the call chain; param1 is the Scroll Bar widget ID.

CAPTION

A caption is a simple widget that shows its descriptor as a string, useful for labeling parts of a window. It always shows its descriptor as its string and is otherwise transparent.

xpWidgetClass_Caption

```
#define xpWidgetClass_Caption 6
```

Caption Properties

`xpProperty_CaptionLit` 1600 This property specifies whether the caption is lit; use lit captions against screens.

GENERAL GRAPHICS

The general graphics widget can show one of many icons available from x-plane.

xpWidgetClass_GeneralGraphics

```
#define xpWidgetClass_GeneralGraphics 7
```

General Graphics Types Values

These define the icon for the general graphics.

<code>xpShip</code>	4
<code>xpILSGlideScope</code>	5
<code>xpMarkerLeft</code>	6
<code>xp_Airport</code>	7
<code>xpNDB</code>	8
<code>xpVOR</code>	9
<code>xpRadioTower</code>	10
<code>xpAircraftCarrier</code>	11
<code>xpFire</code>	12
<code>xpMarkerRight</code>	13
<code>xpCustomObject</code>	14
<code>xpCoolingTower</code>	15
<code>xpSmokeStack</code>	16
<code>xpBuilding</code>	17
<code>xpPowerLine</code>	18
<code>xpVORWithCompassRose</code>	19

xpOilPlatform	21
xpOilPlatformSmall	22
xpWayPoint	23

General Graphics Properties

[xpProperty_GeneralGraphicsType](#) 1700 This property controls the type of icon that is drawn.

PROGRESS INDICATOR

This widget implements a progress indicator as seen when x-plane starts up.

xpWidgetClass_Progress

```
#define xpWidgetClass_Progress 8
```

Progress Indicator Properties

- [xpProperty_ProgressPosition](#) 1800 This is the current value of the progress indicator.
- [xpProperty_ProgressMin](#) 1801 This is the minimum value, equivalent to 0% filled.
- [xpProperty_ProgressMax](#) 1802 This is the maximum value, equivalent to 100% filled.

Pages in category "XPStandardWidgets"

The following 11 pages are in this category, out of 11 total.

- B
 - [Button Properties](#)
- C
 - [Caption Properties](#)
- S
 - [Scroll Bar Properties](#)
- X
 - [XpWidgetClass Button](#)
- X cont.
 - [XpWidgetClass Caption](#)
 - [XpWidgetClass GeneralGraphics](#)
 - [XpWidgetClass MainWindow](#)
 - [XpWidgetClass Progress](#)
- X cont.
 - [XpWidgetClass ScrollBar](#)
 - [XpWidgetClass SubWindow](#)
 - [XpWidgetClass TextField](#)

X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)
 [Log in](#)

Category:XPUIGraphics

(Redirected from [XPUIGraphics](#))

Category: Documentation

[Documentation](#)

XPUIGraphics

UI GRAPHICS

XPWindowStyle

There are a few built-in window styles in X-Plane that you can use.

Note that X-Plane 6 does not offer real shadow-compositing; you must make sure to put a window on top of another window of the right style to the shadows work, etc. This applies to elements with insets and shadows. The rules are:

Sub windows must go on top of main windows, and screens and list views on top of subwindows. Only help and main windows can be over the main screen.

With X-Plane 7 any window or element may be placed over any other element.

Some windows are scaled by stretching, some by repeating. The drawing routines know which scaling method to use. The list view cannot be rescaled in x-plane 6 because it has both a repeating pattern and a gradient in one element. All other elements can be rescaled.

xpWindow_Help	0 An LCD screen that shows help.
xpWindow_MainWindow	1 A dialog box window.
xpWindow_SubWindow	2 A panel or frame within a dialog box window.
xpWindow_Screen	4 An LCD screen within a panel to hold text displays.
xpWindow_ListView	5 A list view within a panel for scrolling file names, etc.

XPDrawWindow

```
WIDGET_API void XPDrawWindow(
    int inX1,
    int inY1,
    int inX2,
    int inY2,
    XPWindowStyle inStyle);
```

This routine draws a window of the given dimensions at the given offset on the virtual screen in a given style. The window is automatically scaled as appropriate using a bitmap scaling technique (scaling or

repeating) as appropriate to the style.

XPGetWindowDefaultDimensions

```
WIDGET_API void XPGetWindowDefaultDimensions(
    XPWindowStyle inStyle,
    int * outWidth,
    int * outHeight);
/* Can be NULL */
/* Can be NULL */
```

This routine returns the default dimensions for a window. Output is either a minimum or fixed value depending on whether the window is scalable.

XPElementStyle

Elements are individually drawable UI things like push buttons, etc. The style defines what kind of element you are drawing. Elements can be stretched in one or two dimensions (depending on the element). Some elements can be lit.

In x-plane 6 some elements must be drawn over metal. Some are scalable and some are not. Any element can be drawn anywhere in x-plane 7.

Scalable Axis Required Background

xpElement_TextField	6	x metal
xpElement_CheckBox	9	none metal
xpElement_CheckBoxLit	10	none metal
xpElement_WindowCloseBox	14	none window header
xpElement_WindowCloseBoxPressed	15	none window header
xpElement_PushButton	16	x metal
xpElement_PushButtonLit	17	x metal
xpElement_OilPlatform	24	none any
xpElement_OilPlatformSmall	25	none any
xpElement_Ship	26	none any
xpElement_ILSGlideScope	27	none any
xpElement_MarkerLeft	28	none any
xpElement_Airport	29	none any
xpElement_Waypoint	30	none any
xpElement_NDB	31	none any
xpElement_VOR	32	none any
xpElement_RadioTower	33	none any
xpElement_AircraftCarrier	34	none any
xpElement_Fire	35	none any
xpElement_MarkerRight	36	none any
xpElement_CustomObject	37	none any
xpElement_CoolingTower	38	none any
xpElement_SmokeStack	39	none any

xpElement_Building	40	none	any
xpElement_PowerLine	41	none	any
xpElement_CopyButtons	45	none	metal
xpElement_CopyButtonsWithEditingGrid	46	none	metal
xpElement_EditingGrid	47	x, y	metal
xpElement_ScrollBar	48	THIS CAN PROBABLY BE REMOVED	
xpElement_VORWithCompassRose	49	none	any
xpElement_Zoomer	51	none	metal
xpElement_TextFieldMiddle	52	x, y	metal
xpElement_LittleDownArrow	53	none	metal
xpElement_LittleUpArrow	54	none	metal
xpElement_WindowDragBar	61	none	metal
xpElement_WindowDragBarSmooth	62	none	metal

XPDrawElement

```

WIDGET_API void                XPDrawElement(
                                int          inX1,
                                int          inY1,
                                int          inX2,
                                int          inY2,
                                XPElementStyle inStyle,
                                int          inLit);

```

XPDrawElement draws a given element at an offset on the virtual screen in set dimensions. EVEN if the element is not scalable, it will be scaled if the width and height do not match the preferred dimensions; it'll just look ugly. Pass inLit to see the lit version of the element; if the element cannot be lit this is ignored.

XPGetElementDefaultDimensions

```

WIDGET_API void                XPGetElementDefaultDimensions(
                                XPElementStyle inStyle,
                                int *          outWidth,
                                /* Can be NULL */
                                int *          outHeight,
                                /* Can be NULL */
                                int *          outCanBeLit);

```

This routine returns the recommended or minimum dimensions of a given UI element. outCanBeLit tells whether the element has both a lit and unlit state. Pass NULL to not receive any of these parameters.

XPTrackStyle

A track is a UI element that displays a value vertically or horizontally. X-Plane has three kinds of tracks: scroll bars, sliders, and progress bars. Tracks can be displayed either horizontally or vertically; tracks will choose their own layout based on the larger dimension of their dimensions (e.g. they know if they are tall or wide). Sliders may be lit or unlit (showing the user manipulating them).

ScrollBar - this is a standard scroll bar with arrows and a thumb to drag. Slider - this is a simple track with a ball in the middle that can be slid. Progress - this is a progress indicator showing how a long task

is going.

xpTrack_ScrollBar 0 not over metal can be lit can be rotated
xpTrack_Slider 1 over metal can be lit can be rotated
xpTrack_Progress 2 over metal cannot be lit cannot be rotated

XPDrawTrack

```
WIDGET_API void XPDrawTrack(
    int inX1,
    int inY1,
    int inX2,
    int inY2,
    int inMin,
    int inMax,
    int inValue,
    XPTrackStyle inTrackStyle,
    int inLit);
```

This routine draws a track. You pass in the track dimensions and size; the track picks the optimal orientation for these dimensions. Pass in the track's minimum current and maximum values; the indicator will be positioned appropriately. You can also specify whether the track is lit or not.

XPGetTrackDefaultDimensions

```
WIDGET_API void XPGetTrackDefaultDimensions(
    XPTrackStyle inStyle,
    int * outWidth,
    int * outCanBeLit);
```

This routine returns a track's default smaller dimension; all tracks are scalable in the larger dimension. It also returns whether a track can be lit.

XPGetTrackMetrics

```
WIDGET_API void XPGetTrackMetrics(
    int inX1,
    int inY1,
    int inX2,
    int inY2,
    int inMin,
    int inMax,
    int inValue,
    XPTrackStyle inTrackStyle,
    int * outIsVertical,
    int * outDownBtnSize,
    int * outDownPageSize,
    int * outThumbSize,
    int * outUpPageSize,
    int * outUpBtnSize);
```

This routine returns the metrics of a track. If you want to write UI code to manipulate a track, this routine helps you know where the mouse locations are. For most other elements, the rectangle the element is drawn in is enough information. However, the scrollbar drawing routine does some automatic placement; this routine lets you know where things ended up. You pass almost everything you would pass to the

draw routine. You get out the orientation, and other useful stuff.

Besides orientation, you get five dimensions for the five parts of a scrollbar, which are the down button, down area (area before the thumb), the thumb, and the up area and button. For horizontal scrollers, the left button decreases; for vertical scrollers, the top button decreases.

Pages in category "XPUIGraphics"

The following 10 pages are in this category, out of 10 total.

X

- [XPDrawElement](#)
- [XPDrawTrack](#)
- [XPDrawWindow](#)
- [XPElementStyle](#)

X cont.

- [XPGetElementDefaultDimensions](#)
- [XPGetTrackDefaultDimensions](#)
- [XPGetTrackMetrics](#)
- [XPGetWindowDefaultDimensions](#)

X cont.

- [XPTrackStyle](#)
- [XPWindowStyle](#)



X-Plane SDK

[category](#) [discussion](#) [view source](#) [history](#)

 [Log in](#)

Category:XPLMScenery

(Redirected from [XPLMScenery](#))

[Category: Documentation](#)

[Documentation](#)

XPLMScenery

This package contains APIs to interact with X-Plane's scenery system.

Version 2.0

Terrain Y-Testing

The Y-testing API allows you to locate the physical scenery mesh. This would be used to place dynamic graphics on top of the ground in a plausible way or do physics interactions.

The Y-test API works via probe objects, which are allocated by your plugin and used to query terrain. Probe objects exist both to capture which algorithm you have requested (see probe types) and also to cache query information.

Performance guidelines: It is generally faster to use the same probe for nearby points and different probes for different points. Try not to allocate more than "hundreds" of probes at most. Share probes if you need more. Generally, probing operations are expensive, and should be avoided via caching when possible.

Y testing returns a location on the terrain, a normal vectory, and a velocity vector. The normal vector tells you the slope of the terrain at that point. The velocity vector tells you if that terrain is moving (and is in meters/second). For example, if your Y test hits the aircraft carrier deck, this tells you the velocity of that point on the deck.

Note: the Y-testing API is limited to probing the loaded scenery area, which is approximately 300x300 km in X-Plane 9. Probes outside this area will return the height of a 0 MSL sphere.

XPLMProbeType

XPLMProbeType defines the type of terrain probe - each probe has a different algorithm. (Only one type of probe is provided right now, but future APIs will expose more flexible or poewrful or useful probes.

`xplm_ProbeY 0` The Y probe gives you the location of the tallest physical scenery along the Y axis going through the queried point.

XPLMProbeResult

Probe results - possible results from a probe query.

`xplm_ProbeHitTerrain` 0 The probe hit terrain and returned valid values.

`xplm_ProbeError` 1 An error in the API call. Either the probe struct size is bad, or the probe is invalid or the type is mismatched for the specific query call.

`xplm_ProbeMissed` 2 The probe call succeeded but there is no terrain under this point (perhaps it is off the side of the planet?)

XPLMProbeRef

```
typedef void * XPLMProbeRef;
```

An XPLMProbeRef is an opaque handle to a probe, used for querying the terrain.

XPLMProbeInfo_t

XPLMProbeInfo_t contains the results of a probe call. Make sure to set structSize to the size of the struct before using it.

```
typedef struct {
    int structSize;
```

Size of structure in bytes - always set this before calling the XPLM.

```
float locationX;
```

Resulting X location of the terrain point we hit, in local OpenGL coordinates.

```
float locationY;
```

Resulting Y location of the terrain point we hit, in local OpenGL coordinates.

```
float locationZ;
```

Resulting Z location of the terrain point we hit, in local OpenGL coordinates.

```
float normalX;
```

X component of the normal vector to the terrain we found.

```
float normalY;
```

Y component of the normal vector to the terrain we found.

```
float normalZ;
```

Z component of the normal vector to the terrain we found.

floatvelocityX;

X component of the velocity vector of the terrain we found.

floatvelocityY;

Y component of the velocity vector of the terrain we found.

floatvelocityZ;

Z component of the velocity vector of the terrain we found.

intis_wet;

Tells if the surface we hit is water (otherwise it is land).

} XPLMProbeInfo_t;

XPLMCreateProbe

XPLM_API XPLMProbeRefXPLMCreateProbe(XPLMProbeTypeinProbeType);

Creates a new probe object of a given type and returns.

XPLMDestroyProbe

XPLM_API voidXPLMDestroyProbe(XPLMProbeRefinProbe);

Deallocates an existing probe object.

XPLMProbeTerrainXYZ

XPLM_API XPLMProbeResultXPLMProbeTerrainXYZ(XPLMProbeRefinProbe,floatinX,floatinY,floatinZ,XPLMProbeInfo_t *outInfo);

Probes the terrain. Pass in the XYZ coordinate of the probe point, a probe object, and an XPLMProbeInfo_t struct that has its structSize member set properly. Other fields are filled in if we hit terrain, and a probe result is returned.

Object Drawing

The object drawing routines let you load and draw X-Plane OBJ files. Objects are loaded by file path and managed via an opaque handle. X-Plane naturally reference counts objects, so it is important that you

balance every successful call to XPLMLoadObject with a call to XPLMUnloadObject!

Version 2.0

XPLMObjectRef

```
typedef void * XPLMObjectRef;
```

An XPLMObjectRef is a opaque handle to an .obj file that has been loaded into memory.

Version 2.0

XPLMDrawInfo_t

The XPLMDrawInfo_t structure contains positioning info for one object that is to be drawn. Be sure to set structSize to the size of the structure for future expansion.

```
typedef struct {
    int structSize;
```

Set this to the size of this structure!

```
float xi;
```

X location of the object in local coordinates.

```
float yi;
```

Y location of the object in local coordinates.

```
float zi;
```

Z location of the object in local coordinates.

```
float pitch;
```

Pitch in degrees to rotate the object, positive is up.

```
float heading;
```

Heading in local coordinates to rotate the object, clockwise.

```
float roll;
```

Roll to rotate the object.

```
} XPLMDrawInfo_t;
```

Version 2.1

XPLMObjectLoaded_f

```
typedef void (* XPLMObjectLoaded_f)(
                                XPLMObjectRef    inObject,
                                void *           inRefcon);
```

You provide this callback when loading an object asynchronously; it will be called once the object is loaded. Your refcon is passed back. The object ref passed in is the newly loaded object (ready for use) or NULL if an error occurred.

If your plugin is disabled, this callback will be delivered as soon as the plugin is re-enabled. If your plugin is unloaded before this callback is ever called, the SDK will release the object handle for you.

Version 2.0

XPLMLoadObject

```
XPLM_API XPLMObjectRef    XPLMLoadObject(
                                const char *    inPath);
```

This routine loads an OBJ file and returns a handle to it. If X-plane has already loaded the object, the handle to the existing object is returned. Do not assume you will get the same handle back twice, but do make sure to call unload once for every load to avoid "leaking" objects. The object will be purged from memory when no plugins and no scenery are using it.

The path for the object must be relative to the X-System base folder. If the path is in the root of the X-System folder you may need to prepend ./ to it; loading objects in the root of the X-System folder is STRONGLY discouraged - your plugin should not dump art resources in the root folder!

XPLMLoadObject will return NULL if the object cannot be loaded (either because it is not found or the file is misformatted). This routine will load any object that can be used in the X-Plane scenery system.

It is important that the datarefs an object uses for animation already be loaded before you load the object. For this reason it may be necessary to defer object loading until the sim has fully started.

Version 2.1

XPLMLoadObjectAsync

```
XPLM_API void XPLMLoadObjectAsync(
    const char *    inPath,
    XPLMObjectLoaded_f inCallback,
    void *          inRefcon);
```

This routine loads an object asynchronously; control is returned to you immediately while X-Plane loads the object. The sim will not stop flying while the object loads. For large objects, it may be several seconds before the load finishes.

You provide a callback function that is called once the load has completed. Note that if the object cannot be loaded, you will not find out until the callback function is called with a NULL object handle.

There is no way to cancel an asynchronous object load; you must wait for the load to complete and then release the object if it is no longer desired.

Version 2.0

XPLMDrawObjects

```
XPLM_API void XPLMDrawObjects(
    XPLMObjectRef inObject,
    int            inCount,
    XPLMDrawInfo_t * inLocations,
    int            lighting,
    int            earth_relative);
```

XPLMDrawObjects draws an object from an OBJ file one or more times. You pass in the object and an array of XPLMDrawInfo_t structs, one for each place you would like the object to be drawn.

X-Plane will attempt to cull the objects based on LOD and visibility, and will pick the appropriate LOD.

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.

earth_relative 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.

Version 2.0

XPLMUnloadObject

```
XPLM_API void XPLMUnloadObject(
    XPLMObjectRef inObject);
```

This routine marks an object as no longer being used by your plugin. Objects are reference counted: once no plugins are using an object, it is purged from memory. Make sure to call XPLMUnloadObject once for each successful call to XPLMLoadObject.

Version 2.0

Library Access

The library access routines allow you to locate scenery objects via the X-Plane library system. Right now library access is only provided for objects, allowing plugin-drawn objects to be extended using the library system.

XPLMLibraryEnumerator_f

```
typedef void (* XPLMLibraryEnumerator_f)(
    const char * inFilePath,
    void * inRef);
```

An XPLMLibraryEnumerator_f is a callback you provide that is called once for each library element that is located. The returned paths will be relative to the X-System folder.

XPLMLookupObjects

```
XPLM_API int XPLMLookupObjects(
    const char * inPath,
    float inLatitude,
    float inLongitude,
    XPLMLibraryEnumerator_f enumerator,
    void * ref);
```

This routine looks up a virtual path in the library system and returns all matching elements. You provide a callback - one virtual path may match many objects in the library. XPLMLookupObjects returns the number of objects found.

The latitude and longitude parameters specify the location the object will be used. The library system allows for scenery packages to only provide objects to certain local locations. Only objects that are allowed at the latitude/longitude you provide will be returned.

Pages in category "XPLMScenery"

The following 15 pages are in this category, out of 15 total.

X

X cont.

X cont.

- [XPLMCreateProbe](#)
 - [XPLMDestroyProbe](#)
 - [XPLMDrawInfo t](#)
 - [XPLMDrawObjects](#)
 - [XPLMLibraryEnumerator f](#)
- [XPLMLoadObject](#)
 - [XPLMLoadObjectAsync](#)
 - [XPLMLookupObjects](#)
 - [XPLMObjectRef](#)
 - [XPLMProbeInfo t](#)
- [XPLMProbeRef](#)
 - [XPLMProbeResult](#)
 - [XPLMProbeTerrainXYZ](#)
 - [XPLMProbeType](#)
 - [XPLMUnloadObject](#)

X-Plane SDK

[page](#) [discussion](#) [view source](#) [history](#)
 [Log in](#)

XPLM Feature Keys

Category: [Documentation](#)
[Documentation](#)

The XPLM 2.0 API allows plugins to query and enable SDK "features". These features alter the behavior of the SDK. The intention is to allow plugins to opt in to additional capabilities. See the [XPLMPlugin](#) API and [XPLMEnableFeature](#) for more information.

"XPLM_WANTS_REFLECTIONS"

This key enables draw callbacks to your plugin during the reflection calculation phase of X-Plane's drawing. Normally, X-Plane will only call your plugin when drawing the main view. However when this feature is present and set to 1, X-Plane will call your plugin multiple times, for both the reflections and regular drawing. This feature key is available in X-Plane 2.0.

"XPLM_USE_NATIVE_PATHS"

This key changes X-Plane's behavior - when it is off (by default) the SDK uses the legacy OS paths - DOS paths on Windows, and HFS paths on Mac. When this capability is set, the entire SDK runs with Unix paths and dir separators.

[Main page](#) [Community portal](#) [Current events](#) [Recent changes](#) [Random page](#) [Help](#)
[What links here](#) [Related changes](#) [Special pages](#)
[Printable version](#) [Permanent link](#)