
OpenFX

Release 1.4

Contributors to the OpenFX Project

Apr 07, 2024

CONTENTS

1	OpenFX reference	3
2	OpenFX Programming Guide	589
3	OpenFX Release Notes	645
	Index	649

The OpenFX documentation is organized as follows:

- The *programming guide* contains everything to get started with OpenFX to create a new plug-in or host application
- The *reference guide* contains the full reference documentation about the OpenFX protocol and design
- The *release notes* contains release notes, documenting changes in recent OpenFX releases.

This documentation is also [available online](#) and can be downloaded as a [PDF](#) or [HTML zip file](#).

This manual is maintained largely by volunteers.

The [Creative Commons Attribution-ShareAlike 4.0 International License \(CC-BY-SA 4.0\)](#) is used for this manual, which is a free and open license. Though there are certain restrictions that come with this license you may in general freely reproduce it and even make changes to it. However, rather than distribute your own version of this manual, we would much prefer if you would send any corrections or changes to the OpenFX association.

OPENFX REFERENCE

This is a mostly complete reference guide to the OFX image effect plugin architecture. It is a backwards compatible update to the 1.2 version of the API. The changes to the API are listed in an addendum.

1.1 Structure of The OFX and the Image Effect API

1.1.1 The Structure Of The Generic OFX API

OFX is actually several things. At its base it is a generic plug-in architecture which can be used to implement a variety of plug-in APIs. The first such API to be implemented on the core architecture is the OFX Image Effect Plug-in API.

It is all specified using the 'C' programming language. C and C++ are the languages mainly used to write visual effects applications (the initial target for OFX plug-in APIs) and have a very wide adoption across most operating systems with many available compilers. By making the API C, rather than C++, you remove the whole set of problems around C++ symbol mangling between the host and plug-in.

APIs are defined in OFX by only a set of C header files and associated documentation. There are no binary libraries for a plug-in or host to link against.

Hosts rely on two symbols within a plug-in, all other communication is boot strapped from those two symbols. The plug-in has no symbolic dependencies from the host. This minimal symbolic dependency allows for run-time determination of what features to provide over the API, making implementation much more flexible and less prone to backwards compatibility problems.

Plug-ins, via the two exposed symbols, indicate the API they implement, the version of the API, their name, their version and their main entry point.

A host communicates with a plug-in via sending 'actions' to the plug-in's main entry function. Actions are C strings that indicate the specific operation to be carried out. They are associated with sets of properties, which allows the main entry function to behave as a generic function.

A plug-in communicates with a host by using sets of function pointers given it by the host. These sets of function pointers, known as 'suites', are named via a C string and a version number. They are returned on request from the host as pointers within a C struct.

Properties are typed value/name pairs that exist on the various OFX objects and are action argument values to the plug-in's main entry point. They are how a plug-in and host pass individual values back and forth to each other. The property suite, defined inside `ofxProperty.h` is used to do this.

1.1.2 OFX APIs

An OFX plug-in API is a named set of actions, properties and suites to perform some specific set of tasks. The first such API that has been defined on the OFX core is the OFX Image Effect API. The set of actions, properties and suites that constitute the API makes up the major part of this document.

Various suites and actions have been defined for the OFX image effect API, however many are actually quite generic and could be reused by other APIs. The property suite definitely has to be used by all other APIs, while the memory allocation suite, the parameter suite and several others would probably be useful for all other APIs. For example the parameter suite could be re-used to specify user visible parameters to the other APIs.

Several types are common to all OFX APIs, and as such are defined in `ofxCORE.h`. Most objects passed back to a plug-in are generally specified by blind data handles, for example:

```
typedef struct OfxPropertySetStruct *OfxPropertySetHandle
```

Blind data structure to manipulate sets of properties through.

This allows for strong typing on functions but allows the implementation of the object to be hidden from the plug-in.

- *OfxStatus*

Used to define a set of status codes indicating the success or failure of an action or suite function

- *OfxHost*

A C struct that is used by a plug-in to get access to suites from a host and properties about the host

- **OfxStatus() OfxPluginEntryPoint (const char *action, const void *handle, OfxPropertySetHandle inArgs, OfxPropertySetHandle outArgs)**

Entry point for plug-ins.

- `action` ASCII c string indicating which action to take
- `instance` object to which action should be applied, this will need to be cast to the appropriate blind data type depending on the *action*
- `inData` handle that contains action specific properties
- `outData` handle where the plug-in should set various action specific properties

This is how the host generally communicates with a plug-in. Entry points are used to pass messages to various objects used within OFX. The main use is within the *OfxPlugin* struct.

The exact set of actions is determined by the plug-in API that is being implemented, however all plug-ins can perform several actions. For the list of actions consult OFX Actions.

A typedef for functions used as main entry points for a plug-in (and several other objects),

- *OfxPlugin*

A C struct that a plug-in fills in to describe itself to a host.

Several general assumptions have been made about how hosts and plug-ins communicate, which specific APIs are allowed to break. The main is the distinction between...

- Descriptors

Which hosts and plug-ins use to define the general behaviour of an object, e.g. the object used to specify what bit depths an Image Effect Plug-in is willing to accept,

- Instances

Which hosts and plug-ins use to control the behaviour of a specific **live** object.

In most APIs descriptors are typically passed from a host to a plug-in during the *kOfxActionDescribe* action, whilst all other actions are passed an instance, e.g: the object passed to the *kOfxActionCreateInstance* action.

1.1.3 The OFX Image Effect API.

The OFX Image Effect Plug-in API is designed for image effect plug-ins for 2D visual effects. This includes such host applications as compositors, editors, rotoscoping tools and colour grading systems.

At heart the image effect API allows a host to send a plug-in a set of images, state the value of a set of parameters and get a resulting image back. However how it does this is somewhat complicated, as the plug-in and host have to negotiate what kind of images are handled, how they can be processed and much more.

1.1.4 Extending OFX

Since items are named using strings, private extensions to OFX can be done simply by *#define*-ing a new string value in a header that is used by both the host and the plug-in. However, to prevent naming collisions, it is important to prefix your new string with a unique identifier. The recommended format is the reverse domain name format of the developer, for example “uk.co.thefoundry”, followed by the new item name.

This applies to anything which could collide, such as suite names, actions, parameter types, pixel depths, image components, contexts, etc. Function names used inside a suite do not need a prefix.

If a private extension is later suggested as and promoted to a standard part of OFX, the standard name will not use a prefix, and it is often the case that during standardization details of the extension might change.

For details on extending the OFX standard, see [Contributing to OpenFX](<https://github.com/AcademySoftwareFoundation/openfx/blob/main/CONTRIBUTING.md>), [OpenFX Standard Update Process](https://github.com/AcademySoftwareFoundation/openfx/blob/main/STANDARD_PROCESS.md), and [OpenFX Project Governance](<https://github.com/AcademySoftwareFoundation/openfx/blob/main/GOVERNANCE.md>).

1.2 The Generic Core API

This chapter describes how plug-ins are distributed and the core API for loading and identifying image effect plug-ins, and the methods of communications between plug-in and host.

1.2.1 OFX Include Files

The C include files that define an OFX API are all that are needed by a plug-in or host to implement the API. Most include files define a set of independent *suites* which are used by a plug-in to communicate with a host application.

There are two include files that are used with nearly every derived API. These are...

- *ofxCore.h* is used to define the basic communication mechanisms between a host and a plug-in. This includes the way in which a plug-in is defined to a host and how to bootstrap the two way communications. It also has several other basic action and property definitions.
- *ofxProperty.h* specifies the property suite, which is how a plug-in gets and sets values on various objects in a host application.

1.2.2 Identifying and Loading Plug-ins

Plug-ins must implement at least two, and normally three, exported functions for a host to identify the plug-ins and to initiate the bootstrapping of communication between the two.

OfxStatus **OfxSetHost**(const *OfxHost* *host)

First thing host should call.

This host call, added in 2020, is not specified in earlier implementation of the API. Therefore host must check if the plugin implemented it and not assume symbol exists. The order of calls is then: 1) OfxSetHost, 2) OfxGetNumberOfPlugins, 3) OfxGetPlugin The host pointer is only assumed valid until OfxGetPlugin where it might get reset. Plug-in can return kOfxStatFailed to indicate it has nothing to do here, it's not for this Host and it should be skipped silently.

int **OfxGetNumberOfPlugins**(void)

Defines the number of plug-ins implemented inside a binary.

A host calls this to determine how many plug-ins there are inside a binary it has loaded. A function of this type must be implemented in and exported from each plug-in binary.

OfxPlugin ***OfxGetPlugin**(int nth)

Returns the 'nth' plug-in implemented inside a binary.

Returns a pointer to the 'nth' plug-in implemented in the binary. A function of this type must be implemented in and exported from each plug-in binary.

OfxSetHost is the very first function called by the host after the binary has been loaded, if it is implemented by the plugin. It passes an *The OfxHost Struct* struct to the plugin to enable the plugin to decide which effects to expose to the host. COMPAT: this call was introduced in 2020; some hosts and/or plugins may not implement it.

OfxGetNumberOfPlugins is the next function called by the host after the binary has been loaded and **OfxSetHost** has been called. The returned pointer to **OfxGetPlugin** and pointers in the struct do not need to be freed in any way by the host.

1.2.3 The Plug-in Main Entry Point And Actions

Actions are how a host communicates with a plug-in. They are in effect generic function calls. Actions are issued via a plug-in's **mainEntry** function pointer found in its *OfxPlugin struct*. The function signature for the main entry point is

OfxStatus() **OfxPluginEntryPoint** (const char *action, const void *handle, **OfxPropertySetHandle** inArgs, **OfxPropertySetHandle** outArgs)

Entry point for plug-ins.

- **action** ASCII c string indicating which action to take
- **instance** object to which action should be applied, this will need to be cast to the appropriate blind data type depending on the *action*
- **inData** handle that contains action specific properties
- **outData** handle where the plug-in should set various action specific properties

This is how the host generally communicates with a plug-in. Entry points are used to pass messages to various objects used within OFX. The main use is within the *OfxPlugin struct*.

The exact set of actions is determined by the plug-in API that is being implemented, however all plug-ins can perform several actions. For the list of actions consult OFX Actions.

The *OfxStatus* value returned is dependent upon the action being called; however the value *kOfxStatReplyDefault* is returned if the plug-in does not trap the action.

The exact set of actions passed to a plug-in's entry point are dependent upon the API the plug-in implements. However, there exists a core set of generic actions that most APIs would use.

1.2.4 Suites

Suites are how a plug-in communicates back to the host. A suite is simply a set of function pointers in a C struct. The set of suites a host needs to implement is defined by the API being implemented. A suite is fetched from a host via the *OfxHost::fetchSuite()* function. This returns a pointer (cast to `void *`) to the named and versioned set of functions. By using this suite fetching mechanism, there is no symbolic dependency from the plug-in to the host, and APIs can be easily expandable without causing backwards compatibility issues.

If the host does not implement a requested suite, or the requested version of that suite, then it should return NULL.

1.2.5 Sequences of Operations Required to Load a Plug-in

The following sequence of operations needs to be performed by a host before it can start telling a plug-in what to do via its `mainEntry` function.

1. the binary containing the plug-in is loaded,
2. (if implemented by plugin and host): the host calls the plug-in's `OfxSetHost` function
3. the number of plug-ins is determined via the `OfxGetNumberOfPlugins` function,
4. for each plug-in defined in the binary
 1. `OfxGetPlugin` is called,
 2. the `pluginApi` and `apiVersion` of the returned `OfxPlugin` struct are examined,
 3. if the plug-in's API or its version are not supported, the plug-in is ignored and we skip to the next one,
 4. the plug-in's pointer is recorded in a plug-in cache,
 5. an appropriate `OfxHost` struct is passed to the plug-in via `setHost` in the returned `OfxPlugin` struct.

1.2.6 Who Owns The Data?

Objects are passed back and forth across the API, and in general, it is the thing that passes the data that is responsible for destroying it. For example the property set handle in the *OfxHost struct* is managed by the host.

There are a few explicit exceptions to this. For example, when an image effect asks for an image from a host it is passed back a property set handle which represents the image. That handle needs to later be disposed of by an effect, by an explicit function call back to the host. These few exceptions are documented with the suite functions that access the object.

Strings

A special case is made for strings. Strings are considered to be of two types, *value* strings and *label* strings. A label string is any string used by OFX to name a property or type. A value string is generally a string value of a property.

More specifically, a label string is a string passed across the API as one of the following...

- a property label (i.e: the `char*` property argument in the property suites)
- a string argument to a suite function which must be one of a set of predefined set of values e.g: `paramType` argument to `OfxParameterSuiteV1::paramDefine`, but not the name argument)

Label strings are considered to be static constant strings. When passed across the API the host/plugin receiving the string neither needs to duplicate nor free the string, it can simply retain the original pointer passed over and use that in future, as it will not change. A host must be aware that when it unloads a plug-in all such pointers will be invalid, and be prepared to cope with such a situation.

A value string is a string passed across the API as one of the following...

- all value arguments to any of the property suite calls
- any other `char*` argument to any other function.

Value strings have no assumptions made about them. When one is passed across the API, the thing that passed the string retains ownership of it. The thing getting the string is not responsible for freeing that string. The scope of the string's validity is until the next OFX API function is called. For example, within a plugin

```
// pointer to store the returned value of the host name
char *returnedHostName;

// get the host name
propSuite->propGetString(hostHandle, kOfxPropName, 0, &returnedHostName);

// now make a copy of that before the next API call, as it may not be valid
// after it
char *hostName = strdup(returnedHostName);

paramSuite->getParamValue(instance, "myParam", &value);
```

1.3 The OfxHost Struct

The `OfxHost` struct is how a host provides plug-ins with access to the various suites that make up the API they implement, as well as a host property set handle which a plug-in can ask questions of. The `setHost` function in the `OfxPlugin` struct is passed a pointer to an `OfxHost` as the first thing to boot-strapping plug-in/host communication.

The `OfxHost` contains two elements,

- `host` - a property set handle that holds a set of properties which describe the host for the plug-in's API
- `fetchSuite` - a function handle used to fetch function suites from the host that implement the plug-in's API

The host property set handle in the `OfxHost` is not global across all plug-ins defined in the binary. It is only applicable for the plug-in whose `setHost` function was called. Use this handle to fetch things like host application names, host capabilities and so on. The set of properties on an OFX Image Effect host is found in the section *Properties on the Image Effect Host*

The `fetchSuite` function is how a plug-in gets a suite from the host. It asks for a suite by giving the C string corresponding to that suite and the version of that suite. The host will return a pointer to that suite, or NULL if it does not

support it. Please note that a suite cannot be fetched until the very first action is called on the plug-in, which is the load action.

struct **OfxHost**

Generic host structure passed to *OfxPlugin::setHost* function.

This structure contains what is needed by a plug-in to bootstrap its connection to the host.

Public Members

OfxPropertySetHandle **host**

Global handle to the host. Extract relevant host properties from this. This pointer will be valid while the binary containing the plug-in is loaded.

const void ***(*fetchSuite*)**(*OfxPropertySetHandle* host, const char *suiteName, int suiteVersion)

The function which the plug-in uses to fetch suites from the host.

- **host** the host the suite is being fetched from this *must* be the *host* member of the *OfxHost* struct containing *fetchSuite*.
- **suiteName** ASCII string labelling the host supplied API
- **suiteVersion** version of that suite to fetch

Any API fetched will be valid while the binary containing the plug-in is loaded.

Repeated calls to *fetchSuite* with the same parameters will return the same pointer.

It is recommended that hosts should return the same host and suite pointers to all plugins in the same shared lib or bundle.

returns

- NULL if the API is unknown (either the api or the version requested),
- pointer to the relevant API if it was found

1.4 The OfxPlugin Struct

This structure is returned by a plugin to identify itself to the host.

```
typedef struct OfxPlugin {
    const char    *pluginApi;
    int           apiVersion;
    const char    *pluginIdentifier;
    unsigned int  pluginVersionMajor;
    unsigned int  pluginVersionMinor;
    void          (*setHost)(OfxHost *host);
    OfxPluginEntryPoint *mainEntry;
} OfxPlugin;
```

pluginApi

This C string tells the host what API the plug-in implements.

apiVersion

This integer tells the host which version of its API the plug-in implements.

pluginIdentifier

This is the globally unique name for the plug-in.

pluginVersionMajor

Major version of this plug-in, this gets incremented whenever software is changed and breaks backwards compatibility.

pluginVersionMinor

Minor version of this plug-in, this gets incremented when software is changed, but does not break backwards compatibility.

setHost

Function used to set the host pointer (see below) which allows the plug-in to fetch suites associated with the API it implements.

mainEntry

The plug-in function that takes messages from the host telling it to do things.

1.4.1 Interpreting the OfxPlugin Struct

When a host gets a pointer back from `OfxGetPlugin`, it examines the string `pluginApi`. This identifies what kind of plug-in it is. Currently there is only one publicly specified API that uses the OFX mechanism, this is `"OfxImageEffectPluginAPI"`, which is the image effect API being discussed by this book. More APIs may be created at a future date, for example `"OfxImageImportPluginAPI"`. Knowing the type of plug-in, the host then knows what suites and host handles are required for that plug-in and what functions the plug-in itself will have. The host passes a `OfxHost` structure appropriate to that plug-in via its `setHost` function. This allows for the same basic architecture to support different plug-in types trivially.

OFX explicitly versions plug-in APIs. By examining the `apiVersion`, the host knows exactly what set of functions the plug-in is going to supply and what version of what suites it will need to provide. This also allows plug-ins to implement several versions of themselves in the same binary, so it can take advantages of new features in a V2 API, but present a V1 plug-in to older hosts that only support V1.

If a host does not support the given plug-in type, or it does not support the given version it should simply ignore that plug-in.

A plug-in needs to uniquely identify itself to a host. This is the job of `pluginIdentifier`. This null terminated ASCII C string should be unique among all plug-ins, it is not necessarily meant to convey a sensible name to an end user. The recommended format is the reverse domain name format of the developer, for example `"uk.co.thefoundry"`, followed by the developer's unique name for the plug-in. e.g. `"uk.co.thefoundry.F_Kronos"`.

A plug-in (as opposed to the API it implements) is versioned with two separate integers in the `OfxPlugin` struct. They serve two separate functions and are,

- `pluginVersionMajor` flags the functionality contained within a plug-in. Incrementing this number means that you have broken backwards compatibility of the plug-in. More specifically, this means a setup from an earlier version, when loaded into this version, will not yield the same result.
- `pluginVersionMinor` flags the release of a plug-in that does not break backwards compatibility, but otherwise enhances that plug-in. For example, increment this when you have fixed a bug or made it faster.

If a host encounters multiple versions of the same plug-in it should,

- when creating a brand new instance, always use the version of a plug-in with the greatest major and minor version numbers,

- when loading a setup, always use the plug-in with the major version that matches the setup, but has the greatest minor number.

As a more concrete example of versioning: the plug-in identified by “org.wibble:Fred” is initially released as 1.0, However a few months later, wibble.org figure out how to make it faster and release it as 1.1. A year later, Fred can now do automatically what a user once needed to set up five parameters to do, thus making it much simpler to use. However this breaks backwards compatibility as the effect can no longer produce the same output as before, so wibble.org then release this as v2.0.

A user’s host might now have three versions of the Fred plug-in on it, v1.0, v1.1 and v2.0. When creating a new instance of the plug-in, the host should always use v2.0. When loading an old project which has a setup from a v1.x plug-in, it should always use the latest, in this case being v1.1.

Note that plug-ins can change the set of parameters between minor version releases. If a plug-in does so, it should do so in a backwards compatible manner, such that the default value of any new parameter would yield the same results as previously. See the chapter below about parameters.

1.5 Packaging OFX Plug-ins

Where a host application chooses to search for OFX plug-ins, what binary format they are in and any directory hierarchy is entirely up to it. However, it is strongly recommended that the following scheme be followed.

1.5.1 Binary Types

Plug-ins should be distributed in the following formats, depending on the host operating system. . . .

- Microsoft Windows, as “.dll” dynamically linked libraries,
- Apple OSX, as binary bundles,
- LINUX (and other Unix variants), as native dynamic shared objects.

1.5.2 Installation Directory Hierarchy

Each plug-in binary is distributed as a Mac OS X package style directory hierarchy. Note that there are two distinct meanings of ‘bundle’, one referring to a binary file format, the other to a directory hierarchy used to distribute software. We are distributing binaries in a bundle package, and in the case of OSX, the binary is a binary bundle. All the binaries must end with “.ofx”, regardless of the host operating system.

The directory hierarchy is as follows.

- NAME.ofx.bundle
 - Contents
 - * Info.plist
 - * Resources
 - NAME.xml
 - EFFECT_A.png
 - EFFECT_A.svg
 - EFFECT_B.png
 - EFFECT_B.svg

```
· ...
* ARCHITECTURE_A
  · NAME.ofx
* ARCHITECTURE_B
  · NAME.ofx
* ...
* ARCHITECTURE_N
  · NAME.ofx
```

Where...

- Info.plist is relevant for OSX only and needs to be filled in appropriately,
- NAME is the file name you want the installed plug-in to be identified by,
- EFFECT_N.png - is an optional PNG image file image to use as an icon for the effect in the plug-in binary which has a matching `pluginIdentifier` field in the `OfxPlugin` struct,
- EFFECT_N.svg - is an optional scalable vector graphic file to use as an icon for the plug-in in the binary which has a matching `pluginIdentifier` field in the `OfxPlugin` struct,
- ARCHITECTURE is the specific operating system architecture the plug-in was built for, these are currently...
 - MacOS - for Apple Macintosh OS X universal or architecture-specific binaries
 - MacOS-x86-64 - *DEPRECATED*. Formerly used for Intel Macs during the transition from PowerPC.
 - Win32 - for Microsoft Windows (compiled 32 bit)
 - Win64 - for Microsoft Windows (compiled 64 bit)
 - IRIX - for SGI IRIX plug-ins (compiled 32 bit) (*DEPRECATED*)
 - IRIX64 - for SGI IRIX plug-ins (compiled 64 bit) (*DEPRECATED*)
 - Linux-x86 - for Linux on x86 CPUs (compiled 32 bit)
 - Linux-x86-64 - for Linux on x86 CPUs running AMD's 64 bit extensions

Note that not all the above architectures need be supported, only the architectures supported by the host product itself.

MacOS Architectures and Universal Binaries

For MacOS, as of 2024 (OpenFX v1.5), all MacOS plug-ins should be in MacOS, and preferably should be universal binaries (whatever that means at any given point in time, given the hosts a plug-in is trying to support). There will not be a specific arm64 folder.

This is acceptable since MacOS supports multi-architecture binaries, so per-architecture subdirectories are not needed for a host application to find the proper architecture plug-in. This proposal would prohibit a plug-in from shipping a separate Intel and arm64 binaries (since the plug-in .ofx file must always be named to match the top-level bundle name), but it should not be difficult for any plug-in to merge multiple architectures into a single binary on MacOS.

Future Architecture Compatibility

NOTE: this section is not yet normative, but informational.

Note that Windows and Linux do not support universal binaries. Each .ofx file is for one architecture, so a host must check the proper architecture-specific subdir.

When Windows and/or Linux support alternative processor architectures such as arm64, hosts should look in appropriately-named subdirs for the proper .ofx plugin file. On Windows with arm64, Win-arm64 should be used (vs. current Win32 and Win64 which are Intel-specific). On Linux, hosts should look in the subdir named by Linux-`${uname -m}` which for arm64 should be Linux-aarch64. Using `uname -m` rather than a hard-coded list allows for transparently supporting any future architectures.

Structure

This directory structure is necessary on OS X, but it also gives a nice skeleton to hang all other operating systems from in a single install, as well as a clean place to put resources.

The Info.plist is specific to Apple and you should consult the Apple developer's website for more details. It should contain the following keys:

- CFBundleExecutable - the name of the binary bundle in the MacOS directory
- CFBundlePackageType - to be BNDL
- CFBundleInfoDictionaryVersion
- CFBundleVersion
- CFBundleDevelopmentRegion

1.5.3 Installation Location

Plug-ins are searched for in a variety of locations, both default and user specified. All such directories are examined for plug-in bundles and sub directories are also recursively examined.

A list of directories is supplied in the "OFX_PLUGIN_PATH" environment variable, these are examined, first to last, for plug-ins, then the default location is examined.

On Microsoft Windows machines, the plug-ins are searched for in:

1. the ';'-separated directory list specified by the environment variable "OFX_PLUGIN_PATH"
2. the directory returned by `getStdOFXPluginPath` in the following code snippet:

```
#include "shobj.h"
#include "tchar.h"
const TCHAR *getStdOFXPluginPath(void)
{
    static TCHAR buffer[MAX_PATH];
    static int gotIt = 0;
    if(!gotIt) {
        gotIt = 1;
        SHGetFolderPath(NULL, CSIDL_PROGRAM_FILES_COMMON, NULL, SHGFP_TYPE_CURRENT, &
↪buffer);
        _tcscat(buffer, __T("\\OFX\\Plugins"));
    }
}
```

(continues on next page)

```
return buffer;
}
```

3. the directory C:\Program Files\Common Files\OFX\Plugins. This location is deprecated, and it is returned by the code snippet above on English language systems. However it should still be examined by hosts for backwards compatibility.

On Apple OSX machines, the plug-ins are searched for in:

1. the ‘;’-separated directory list specified by the environment variable “OFX_PLUGIN_PATH”
2. the directory /Library/OFX/Plugins

On UNIX, Linux and other UNIX like operating systems, the plug-ins are searched for in:

1. the ‘:’-separated directory specified by the environment variable “OFX_PLUGIN_PATH”
2. the directory /usr/OFX/Plugins

Any bundle or sub-directory name starting with the character ‘@’ is to be ignored. Such directories or bundles must be skipped by the host.

1.5.4 Plug-in Icons

Some hosts may wish to display an icon associated with the effects in a plug-in binary on their interfaces. Any such icon must be in the Portable Network Graphics format (see <http://www.libpng.org/>) and must contain 32 bits of colour, including an alpha channel. Ideally it should be at least 128x128 pixels.

Note that a single plug-in binary may define more than one effect, when *OfxGetNumberOfPlugins* returns a value greater than 1. These icons are specific to each effect within the plug-in, and are named according to what is returned from *OfxGetPlugin*.

Host applications should dynamically resize the icon to fit their preferred icon size. The icon should not have its aspect changed, rather the host should fill with some appropriate colour any blank areas due to aspect mismatches.

Ideally plug-in developers should not render the plug-in or effect’s name into the icon, as this may be changed by the resource file, especially for internationalisation purposes. Hosts should thus present the plug-in and/or effect’s name next to the icon in some way.

The icon file must be named as the corresponding `pluginIdentifier` field from the `OfxPlugin`, postpended with ‘.png’ and be placed in the resources sub-directory.

Some hosts may use a scalable vector icon if provided; it should be in SVG format and be named and located just like the .png icon but with a .svg suffix.

1.5.5 Externally Specified Resources

Some plug-ins may supply an externally specified resource file for particular hosts. Typically this is for tasks such as internationalising interfaces, tweaking user interfaces for specific hosts, and so on. These are XML files and have DTD associated with the specific API, for example OFX Image Effect DTD is found in `ofx.dtd`.

The XML resource file is installed in the Resources subdirectory of the bundle hierarchy. Its name should be `NAME.xml`, where name is the base name of the bundle folder and the effect binary.

Plug-ins are free to include other resources in the Resources subdirectory.

1.6 The Image Effect API

1.6.1 Introduction

In general, image effects plug-ins take zero or more input clips and produce an output clip. So far so simple, however there are many devils hiding in the details. Several supporting suites are required from the host and the plug-in needs to respond to a range of actions to work correctly. How an effect is intended to be used also complicates the issue, forcing sets of behaviours depending on the context of an effect.

Plug-ins that implement the image effect API set the `pluginApi` member of the *OfxPlugin struct* returned by the global *OfxGetPlugin* to be:

kOfxImageEffectPluginApi

String used to label OFX Image Effect Plug-ins.

Set the `pluginApi` member of the *OfxPluginHeader* inside any *OfxImageEffectPluginStruct* to be this so that the host knows the plug-in is an image effect.

The current version of the API is 1. This is enough to label the plug-in as an image effect plug-in.

1.6.2 Image Effect API Header Files

The header files used to define the OFX Image Effect API are...

- [ofxCore.h](#) Provides the core definitions of the general OFX architecture that allow the bootstrapping of specific APIs, as well as several core actions,
- [ofxProperty.h](#)
Provides generic property fetching suite used to get and set values about objects in the API,
- [ofiParam.h](#) Provides the suite for defining user visible parameters to an effect
- [ofxMultiThread.h](#) Provides the suite for basic multi-threading capabilities
- [ofxInteract.h](#) Provides the suite that allows a plug-in to use OpenGL to draw their own interactive GUI tools
- [ofxKeySyms.h](#) Provides key symbols used by 'Interacts' to represent keyboard events
- [ofxMemory.h](#) Provides a simple memory allocation suite,
- [ofxMessage.h](#) Provides a simple messaging suite to communicate with an end user
- [ofxImageEffect.h](#) Defines a suite and set of actions that draws all the above together to create an visual effect plug-in.
- [ofxDrawSuite.h](#) Provides an optional suite that allows a plug-in to draw their own interactive GUI tools without using OpenGL

These contain the suite definitions, property definitions and action definitions that are used by the API.

1.6.3 Actions Used by the API

All image effect plug-ins have a main entry point. This is used to trap all the standard actions used to drive the plug-in. They can also have other optional entry points that allow the plug-in to create custom user interface widgets. These *interact* entry points are specified during the two description actions.

The following actions can be passed to a plug-in's main entry point...

- The Generic Load Action called just after a plug-in is first loaded,
- The Generic Unload Action called just before a plug-in is unloaded,
- The Generic Describe Action called to describe a plug-in's behaviour to a host,
- The Generic Create Instance Action called just after an instance is created,
- The Generic Destroy Instance Action called just before an instance is destroyed,
- The Generic Begin/End Instance Changed Actions , a pair of actions used to bracket a set of Instance Changed actions,
- The Generic Instance Changed Action an action used to indicate that a value has changed in a plug-in instance,
- The Generic Purge Caches Action called to have the plug-in delete any temporary private data caches it may have,
- The Sync Private Data Action called to have the plug-in sync any private state data back to its data set,
- The Generic Begin/End Instance Edit Actions a pair of calls that are used to bracket the fact that a user interface has been opened on an instance and it is being edited,
- The Begin Sequence Render Action where a plug-in is told that it is about to render a sequence of images,
- The Render Action where a plug-in is told that it is to render an output image,
- The End Sequence Render Action where a plug-in is told it has finished rendering a sequence of images,
- The Describe In Context Action used to have a plug-in describe itself in a specific context,
- The Get Region of Definition Action where an instance gets to state how big an image it can create,
- The Get Regions Of Interest Action where an instance gets to state how much of its input images it needs to create a give output image,
- The Get Frames Needed Action where an instance gets to state how many frames of input it needs on a given clip to generate a single frame of output,
- The Is Identity Action where an instance gets to state that its current state does not affect its inputs, so that the output can be directly copied from an input clip,
- The Get Clip Preferences Action where an instance gets to state what data and pixel types it wants on its inputs and will generate on its outputs,
- The Get Time Domain Action where a plug-in gets to state how many frames of data it can generate.

1.6.4 Main Objects Used by the API

The image effect API uses a variety of different objects. Some are defined via blind data handles, others via property sets, and some by a combination of the two. These objects are...

- Host Descriptor - a descriptor object used by a host to describe its behaviour to a plug-in,
- Image Effect Descriptor - a descriptor object used by a plug-in to describe its behaviour to a host,
- Image Effect Instance - an instance object maintaining state about an image effect,
- Clip Descriptor - a descriptor object for a sequence of images used as input or output a plug-in may use,
- Clip Instance - a instance object maintaining state about a sequence of images used as input or output to an effect instance,
- Parameter Descriptor - a descriptor object used to specify a user visible parameter in an effect descriptor,
- Parameter Instance - an instance object that maintains state about a user visible parameter in an effect instance,
- Parameter Set Descriptor - a descriptor object used to specify a set of user visible parameters in an effect descriptor,
- Parameter Set Instance - an instance object that maintains state about a set of user visible parameters in an effect instance,
- Image Instance - a instance object that maintains state about a 2D image being passed to an effect instance.
- Interact Descriptor - which describes a custom openGL user interface, for example an overlay over the inputs to an image effect. These have a separate entry point to an image effect.
- Interact Instance - which holds the state on a custom openGL user interface. These have a separate entry point to an image effect.

Host Descriptors

The host descriptor is represented by the properties found on the host property set handle in the *OfxHost struct*. The complete set of read only properties are found in the section Properties on the Image Effect Host.

These sets of properties are there to describe the capabilities of the host to a plug-in, thus giving a plug-in the ability to modify its behaviour depending on the capabilities of the host.

A host descriptor is valid while a plug-in is loaded.

Effects

An effect is an object in the OFX Image Effect API that represents an image processing plug-in. It has associated with it a set of properties, a set of image clips and a set of parameters. These component objects of an effect are defined and used by an effect to do whatever processing it needs to. A handle to an image effect (instance or descriptor) is passed into a plug-in's *main entry point handle* argument:

```
typedef struct OfxImageEffectStruct *OfxImageEffectHandle
```

Blind declaration of an OFX image effect.

The functions that directly manipulate an image effect handle are specified in the *OfxImageEffectSuiteV1* found in the header file *ofxImageEffect.h*.

Effect Descriptors

An effect descriptor is an object of type *OfxImageEffectHandle* passed into an effect's *main entry point handle* argument. The two actions it is passed to are:

- *kOfxActionDescribe*
- *kOfxImageEffectActionDescribeInContext*

An effect descriptor does not refer to a 'live' effect, it is a handle which the effect uses to describe itself back to the host. It does this by setting a variety of properties on an associated property handle, and specifying a variety of objects (such as clips and parameters) using functions in the available suites.

Once described, a host should cache away the description in some manner so that when an instance is made, it simply looks at the description and creates the necessary objects needed by that instance. This stops the overhead of having every instance be forced to describe itself over the API.

Effect descriptors are only valid in an effect for the duration of the instance they were passed into.

The properties on an effect descriptor can be found in the section Properties on an Effect Descriptor.

Effect Instances

An effect instance is an object of type *OfxImageEffectHandle* passed into an effect's *main entry point handle* argument. The *handle* argument should be statically cast to this type. It is passed into all actions of an image effect that a descriptor is not passed into.

The effect instance represents a 'live' instance of an effect. Because an effect has previously been described, via an effect descriptor, an instance does not have to respecify the parameters, clips and properties that it needs. This means, that when an instance is passed to an effect, all the objects previously described will have been created.

Generally multiple instances of an effect can be in existence at the same time, each with a different set of parameters, clips and properties.

Effect instances are valid between the calls to *kOfxActionCreateInstance* and *kOfxActionDestroyInstance*, for which it is passed as the *handle* argument.

The properties on an effect instance can be found in the section Properties on an Effect Instance.

Clips

A clip is a sequential set of images attached to an effect. They are used to fetch images from a host and to specify how a plug-in wishes to manage the sequence.

Clip Descriptors

Clip descriptors are returned by the *OfxImageEffectSuiteV1::clipDefine()* function. They are used during the *kOfxActionDescribe* action by an effect to indicate the presence of an input or output clip and how that clip behaves.

A clip descriptor is only valid for the duration of the action it was created in.

The properties on a clip descriptor can be found in the section Properties on a Clip Descriptor.

Clip Instances

```
typedef struct OfxImageClipStruct *OfxImageClipHandle
```

Blind declaration of an OFX image effect.

Clip instances are returned by the *OfxImageEffectSuiteV1::clipGetHandle()* function. They are used to access images and manipulate properties on an effect instance's input and output clips. A variety of functions in the *OfxImageEffectSuiteV1* are used to manipulate them.

A clip instance is valid while the related effect instance is valid.

The properties on a clip instance can be found in the section Properties on a Clip Instance.

Parameters

Parameters are user visible objects that an effect uses to specify its state, for example a floating point value used to control the blur size in a blur effect. Parameters (both descriptors and instances) are represented as blind data handles of type:

```
typedef struct OfxParamStruct *OfxParamHandle
```

Blind declaration of an OFX param.

Parameter sets are the collection of parameters that an effect has associated with it. They are represented by the type *OfxParamSetHandle*. The contents of an effect's parameter set are defined during the *kOfxImageEffectActionDescribeInContext* action. Parameters cannot be dynamically added to, or deleted from an effect instance.

Parameters can be of a wide range of types, each of which have their own unique capabilities and property sets. For example a colour parameter differs from a boolean parameter.

Parameters and parameter sets are manipulated via the calls and properties in the *OfxParameterSuiteV1* specified in *ofxParam.h*. The properties on parameter instances and descriptors can be found in the section Properties on Parameter Descriptors and Instances.

Parameter Set Descriptors

Parameter set descriptors are returned by the `cpp:func`OfxImageEffectSuiteV1::getParamSet`` function. This returns a handle associated with an image effect descriptor which can be used by the parameter suite routines to create and describe parameters to a host.

A parameter set descriptor is valid for the duration of the *kOfxImageEffectActionDescribeInContext* action in which it is fetched.

Parameter Descriptors

Parameter descriptors are returned by the *OfxParameterSuiteV1::paramDefine()* function. They are used to define the existence of a parameter to the host, and to set the various attributes of that parameter. Later, when an effect instance is created, an instance of the described parameter will also be created.

A parameter descriptor is valid for the duration of the *kOfxImageEffectActionDescribeInContext* action in which it is created.

Parameter Set Instances

Parameter set instances are returned by the `OfxImageEffectSuiteV1::getParamSet()` function. This returns a handle associated with an image effect instance which can be used by the parameter suite routines to fetch and describe parameters to a host.

A parameter set handle instance is valid while the associated effect instance remains valid.

Parameter Instances

Parameter instances are returned by the `OfxParameterSuiteV1::paramGetHandle()` function. This function fetches a previously described parameter back from the parameter set. The handle can then be passed back to the various functions in the `OfxParameterSuiteV1` to manipulate it.

A parameter instance handle remains valid while the associated effect instance remains valid.

Image Instances

An image instance is an object returned by the `OfxImageEffectSuiteV1::clipGetImage()` function. This fetches an image out of a clip and returns it as a property set to the plugin. The image can be accessed by looking up the property values in that set, which includes the data pointer to the image.

An image instance is valid until the effect calls `OfxImageEffectSuiteV1::clipReleaseImage()` on the property handle. The effect *must* release all fetched images before it returns from the action.

The set of properties that make up an image can be found in the section Properties on an Image.

Interacts

An interact is an OFX object that is used to draw custom user interface elements, for example overlays on top of a host's image viewer or custom parameter widgets. Interacts have their own *main entry point*, which is separate to the effect's main entry point. Typically an interact's main entry point is specified as a pointer property on an OFX object, for example the `kOfxImageEffectPluginPropOverlayInteractV1` property on an effect descriptor.

The functions that directly manipulate interacts are in the `OfxInteractSuiteV1` found in the header file `ofxInteract.h`, as well as the properties and specific actions that apply to interacts.

Interact Descriptors

Interact descriptors are blind handles passed to the `kOfxActionDescribeInteract` sent to an interact's separate main entry point. They should be cast to the type `OfxInteractHandle`.

The properties found on a descriptor are found in section Properties on Interact Descriptors.

Interact Instances

Interact instances are blind handles passed to all actions but the *kOfxActionDescribe* sent to an interact's separate main entry point. They should be cast to the type

```
typedef struct OfxInteract *OfxInteractHandle
```

Blind declaration of an OFX interactive gui.

The properties found on an instance are found in section Properties on Interact Instance.

1.7 Image Processing Architectures

OFX supports a range of image processing architectures. The simpler ones being special cases of the most complex one. Levels of support, in both plug-in and host, are signalled by setting appropriate properties in the plugin and host.

This chapter describes the most general architecture that OFX can support, with simpler cases just being specialisations of the general case.

1.7.1 The Image Plane

At it's most generalised, OFX allows for a complex imaging architecture based around an infinite 2D plane on which we are filling in pixels.

Firstly, there is some subsection of this infinite plane that the user wants to be the end result of their work, call this the project extent. The project extent is always rooted, on its bottom left, at the origin of the image plane. The project extent defines the upper right hand corner of the project window. For example a PAL sized project spans (0, 0) to (768, 576) on the image plane.

We define an image effect as something that can fill in a rectangle of pixels in this infinite plane, possibly using images defined at other locations on this image plane.

1.7.2 Regions of Definition

An effect has a **Region of Definition** (RoD), this is is the maximum area of the plane that the effect can fill in. for example: a 'read source media' effect would only be able to fill an area as big as it's source media. An effect's RoD may need to be based on the RoD of its inputs, for example: the RoD of a contrast/brightness colour corrector would generally be the RoD of it's input, while the RoD of a rotation effect would be bigger than that of it's input image.

The purpose of the *kOfxImageEffectActionGetRegionOfDefinition* action is for the host to ask an effect what its region of definition is. An effect calculates this by looking at its input clips and the values of its current parameters.

Hosts are not obliged to render all an effects RoD, as it may have fixed frame sizes, or any number of other issues.

Infinite RoDs

Infinite RoDs are used to indicate an effect can fill pixels in anywhere on the image plane it is asked to. For example a no-input noise generator that generates random colours on a per pixel basis. An infinite RoD is flagged by setting the minimums to be:

`kOfxFlagInfiniteMin`

Used to flag infinite rects. Set minimums to this to indicate infinite.

This is effectively `INT_MIN`

and the maximums to be:

`kOfxFlagInfiniteMax`

Used to flag infinite rects. Set maximums to this to indicate infinite.

This is effectively `INT_MAX`.

for both double and integer rects. Hosts and plug-ins need to be infinite RoD aware. Hosts need to clip such RoDs to an appropriate rectangle, typically the project extent. Plug-ins need to check for infinite RoDs when asking input clips for them and to pass them through unless they explicitly clamp them. To indicate an infinite RoD set it as indicated in the following code snippet.

```
outputRoD.x1 = kOfxFlagInfiniteMin;
outputRoD.y1 = kOfxFlagInfiniteMin;
outputRoD.x2 = kOfxFlagInfiniteMax;
outputRoD.y2 = kOfxFlagInfiniteMax;
```

1.7.3 Regions Of Interest

An effect will be asked to fill in some region of this infinite plane. The section it is being asked to fill in is called the **Region of Interest** (RoI).

Before an effect has been asked to process a given RoI, it will be asked to specify the area of each input clip it will need to process that area. For example: a simple colour correction effect only needs as much input as it does output, while a blur will need an area that is larger than the specified RoI by a border of the same width as the blur radius.

The purpose of the `kOfxImageEffectActionGetRegionsOfInterest` action is for the host to ask an effect what areas it needs from each input clip, to render a specific output region. An effect needs to examine its set of parameters and the region it has been asked to render to determine how much of *each* input clip it needs.

1.7.4 Tiled Rendering

Tiling is the ability of an effect to manage images that are less than full frame (or in our current nomenclature, less than the full Region of Definition). By tiling the images it renders, a host will render an effect in several passes, say by doing the bottom half, then the top half.

Hosts may tile rendering for a variety of reasons. Usually it is in an attempt to reduce memory demands or to distribute rendering of an effect to several different CPUs or computers.

Effects that in effect only perform per pixel calculations (for example a simple colour gain effect) tile very easily. However in the most general case for effects, tiling may be self defeating, as an effect, in order to render a tile, may need significantly more from its input clips than the tile in question. For example, an effect that performs a 2D transform on its input image, may need to sample all that image even when rendering a very small tile on output, as the input image may have been scaled down so that it only covers a few pixels on output.

1.7.5 Tree Based Architectures

The most general compositing hosts allow images to be of any size at any location on our image plane. They also plumb the output of effects into other effects, to create effect trees. When evaluating this tree of effects, a general host will want to render the minimum number of pixels it needs to fill in the final desired image. Typically the top level of this compositing tree is being rendered at a certain project size, for example PAL SD, 2K film and so on. This is where the RoD/RoI calls come in handy.

The host asks the top effect how much picture information it can produce, which in turn asks effects below it their RoDs and so on until leaf effects are reached, which report back up the tree until the top effect calculates its RoD and reports back to the host. The host typically clips that RoD to its project size.

Having determined in this way the window it wants rendered at the top effect, the host asks the top node the regions of interest on each of its inputs. This again propagates down the effect tree until leaf nodes are encountered. These regions of interest are cached at effect for later use.

At this point the host can start rendering, from the bottom of the tree upwards, by asking each effect to fill in the region of interest that was previously specified in the RoI walk. These regions are then passed to the next level up to render and so on.

Another complication is tiling. If a host tiles, it will need to walk the tree and perform the RoI calculation for each tile that it renders.

The details may differ on specific hosts, but this is more or less the most generic way compositing hosts currently work.

1.7.6 Simpler Architectures

The above architecture is quite complex, as the inputs supplied can lie anywhere on the image plane, as can the output, and they can be subsections of the 'complete' image. Not all hosts work in this way, generally it is only the more advance compositing systems working on large resolution images.

Some other systems allow for images to be anywhere on the image plane, but always pass around full RoD images, never tiles.

The simplest systems, don't have any of the above complexity. The RoDs, RoIs, images and project sizes in such systems are exactly the same, always. Often these are editing, as opposed to compositing, systems.

Similarly, some plugin effects cannot handle sub RoD images, or even images not rooted at the origin.

The OFX architecture is meant to support all of them. Assuming a plugin supports the most general architecture, it will trivially run on hosts with simpler architectures. However, if a plugin does not support tiled, or arbitrarily positioned images, they may not run cleanly on hosts that expect them to do so.

To this end, two properties are provided that flag the capabilities of a plugin or host...

- *kOfxImageEffectPropSupportsMultiResolution* which indicates support for images of differing sizes not centred on the origin,
- *kOfxImageEffectPropSupportsTiles* which indicates support for images that contain less than full frame pixel data

A plug-in should flag these appropriately, so that hosts know how to deal with the effect. A host can either choose to refuse to load a plugin, or, preferentially, pad images with an appropriate amount of black/transparent pixels to enable them to work.

The *kOfxImageEffectActionGetRegionsOfInterest* is redundant for plugins that do not support tiled rendering, as the plugin is asking that it be given the full Region of Definition of all its inputs. A host may have difficulty doing this (for example with an input that is attached to an effect that can create infinite images such as a random noise generator), if so, it should clamp images to some a size in some manner.

The RoD/RoI actions are potentially redundant on simpler hosts. For example fixed frame size hosts. If a host has no need to call these actions, it simply should not.

1.8 Image Effect Contexts

How an image effect is used by an end user affects how it should interact with a host application. For example an effect that is to be used as a transition between two clips works differently to an effect that is a simple filter. One must have two inputs and know how much to mix between the two input clips, the other has fewer constraints on it. Within OFX we have standardised several different uses and have called them *contexts*.

More specifically, a context mandates certain behaviours from an effect when it is described or instantiated in that context. The major issue is the number of input clips it takes, and how it can interact with those input clips.

All OFX contexts have a single output clip and zero or more input clips. The current contexts defined in OFX are:

- *kOfxImageEffectContextGenerator*

No compulsory input clips used by a host to create imagery from scratch, e.g: a noise generator

- *kOfxImageEffectContextFilter*

A single compulsory input clip. A traditional ‘filter effect’ that transforms a single input in some way, e.g: a simple blur

- *kOfxImageEffectContextTransition*

Two compulsory input clips and a compulsory ‘Transition’ double parameter Used to perform transitions between clips, typically in editing applications, eg: a cross dissolve,

- *kOfxImageEffectContextPaint*

Two compulsory input clips, one image to paint onto, the other a mask to control where the effect happens Used by hosts to use an effect under a paint brush

- *kOfxImageEffectContextRetimer*

A single compulsory input clip, and a compulsory ‘SourceTime’ double parameter Used by a host to change the playback speed of a clip,

- *kOfxImageEffectContextGeneral*

An arbitrary number of inputs, generally used in a ‘tree’ compositing environment, a catch all context.

A host or plug-in need not support all contexts. For example a host that does not have any paint facility within it should not need to support the paint context, or a simple blur effect need not support the retimer context.

An effect may say that it can be used in more than one context, for example a blur effect that acts as a filter, with a single input to blur, and a general effect, with an input to blur and an optional input to act as a mask to attenuate the blur. In such cases a host should choose the most appropriate context for the way that host’s architecture. With our blur example, a tree based compositing host should simply ignore the filter context and always use it in the general context.

Plugins and hosts inform each other what contexts they work in via the multidimensional *kOfxImageEffectPropSupportedContexts* property.

A host indicates which contexts it supports by setting the *kOfxImageEffectPropSupportedContexts* property in the global host descriptor. A plugin indicates which contexts it supports by setting this on the effect descriptor passed to the *kOfxActionDescribe* action.

Because a plugin can work in different ways, it needs the ability to describe itself to the host in different ways. This is the purpose of the *kOfxImageEffectActionDescribeInContext* action. This action is called once for each context that the effect supports, and the effect gets to describe the input clips and parameters appropriate to that context. This means that an effect can have different sets of parameters and clips in different contexts, though it will most likely have

a core set of parameters that it uses in all contexts. From our blur example, both the filter and general contexts would have a ‘blur radius’ parameter, but the general context might have an ‘invert matte’ parameter.

During the *kOfxImageEffectActionDescribeInContext* action, an effect must describe all clips and parameters that it intends to use. This includes the mandated clips and parameters for that context.

A plugin instance is created in a specific context which will not be changed over the lifetime of that instance. The context can be retrieved from the instance via the *kOfxImageEffectPropContext* property on the instance handle.

1.8.1 The Generator Context

A generator context is for cases where a plugin can create images without any input clips, eg: a colour bar generator.

In this context, a plugin has the following mandated clips,

- an output clip named *Output*

Any input clips that are specified must be optional.

A host is responsible for setting the initial preferences of the output clip, it must do this in a manner that is transparent to the plugin. So the pixel depths, components, fielding, frame rate and pixel aspect ratio are under the control of the host. How it arrives at these is a matter for the host, but as a plugin specifies what components it can produce on output, as well as the pixel depths it supports, the host must choose one of these.

Generators still have Regions of Definition. This should generally be,

- based on the project size eg: an effect that renders a 3D sky simulation,
- based on parameter settings eg: an effect that renders a circle in an arbitrary location,
- infinite, which implies the effect can generate output anywhere on the image plane.

The pixel preferences action is constrained in this context by the following,

- a plugin cannot change the component type of the *Output* clip,

1.8.2 The Filter Context

A filter effect is the ordinary way most effects are used with a single input. They allow track or layer based hosts that cannot present extra input to use an effect.

In this context, a plugin has the following mandated objects...

- an input clip named *Source*
- an output clip named *Output*

Other input clips may be described, which must all be optional. However there is no way to guarantee that all hosts will be able to wire in such clips, so it is suggested that in cases where effects can take single or multiple inputs, they expose themselves in the filter context with a single input and the general context with multiple inputs.

The pixel preferences action is constrained in this context by the following,

- a plugin cannot change the component type of the *Output* clip, it will always be the same as the *Source* clip,

1.8.3 The Transition Context

Transitions are effects that blend from one clip to another over time, eg: a wipe or a cross dissolve.

In this context, a plugin has the following mandated objects...

- an input clip names 'SourceFrom'
- an input clip names 'SourceTo'
- an output clip named *Output*
- a single double parameter called 'Transition' (see Mandated Parameters)

Any other input clips that are specified must be optional. Though it is suggested for simplicity's sake that only the two mandated clips be used.

The 'Transition' parameter cannot be labelled, positioned or controlled by the plug-in in anyway, it can only have it's value read, which will have a number returned between the value of 0 and 1. This number indicates how far through the transition the effect is, at 0 it should output all of 'SourceFrom', at 1 it should output all of 'SourceTo', in the middle some appropriate blend.

The pixel preferences action is constrained in this context by the following,

- the component types of the "SourceFrom", "SourceTo" and *Output* clips will always be the same,
- the pixel depths of the "SourceFrom", "SourceTo" and *Output* clips will always be the same,
- a plugin cannot change any of the pixel preferences of any of the clips.

1.8.4 The Paint Context

Paint effects are effects used inside digital painting system, where the effect is limited to a small area of the source image via a masking image. Perhaps 'brush' would have been a better choice for the name of the context.

In this context, a plugin has the following mandated objects...

- an input clip names *Source*,
- an input clip names *Brush*, the only component type it supports is 'alpha',
- an output clip named *Output*.

Any other input clips that are specified must be optional.

The masking images consists of pixels from 0 to the white point of the pixel depth. Where the mask is zero the effect should not occur, where the effect is whitepoint the effect should be 'full on', where it is grey the effect should blend with the source in some manner.

The masking image may be smaller than the source image, even if the effect states that it cannot support multi-resolution images.

The pixel preferences action is constrained in this context by the following,

- the pixel depths of the *Source*, *Brush* and *Output* clips will always be the same,
- the component type of *Source* and *Output* will always be the same,
- a plugin cannot change any of the pixel preferences of any of the clips.

1.8.5 The Retimer Context

The retimer context is for effects that change the length of a clip by interpolating frames from the source clip to create an inbetween output frame.

In this context, a plugin has the following mandated objects...

- an input clip names *Source*
- an output clip named *Output*
- a 1D double parameter named 'SourceTime' (see Mandated Parameters)

Any other input clips that are specified must be optional.

The 'SourceTime' parameter cannot be labelled, positioned or controlled by the plug-in in anyway, it can only have it's value read. Its value is how the source time to maps to the output time. So if the output time is '3' and the 'SourceTime' parameter returns 8.5 at this time, the resulting image should be an interpolated between source frames 8 and 9.

The pixel preferences action is constrained in this context by the following,

- the pixel depths of the *Source* and *Output* clips will always be the same,
- the component type of *Source* and *Output* will always be the same,
- a plugin cannot change any of the pixel preferences of any of the clips.

1.8.6 The General Context

The general context is to some extent a catch all context, but is generally how a 'tree' effect should be instantiated. It has no constraints on its input clips, nor on the pixel preferences actions.

In this context, has the following mandated objects...

- an output clip named *Output*

1.8.7 Parameters Mandated In A Context

The retimer and transition context both mandate a parameter be declared, the double params 'SourceTime' and 'Transition'. The purpose of these parameters is for the host to communicate with the plug-in, they are *not* meant to be treated as normal parameters, exposed on the user plug-in's user interface.

For example, the purpose of a transition effect is to dissolve in some interesting way between two separate clips, under control of the host application. Typically this is done on systems that edit. The mandated 'Transition' double pseudo-parameter is not a normal one exposed on the plug-in UI, rather it is the way the host indicates how far through the transition the effect is. For example, think about two clips on a time line based editor with a transition between them, the host would set the value value of the 'Transition' parameter implicitly by how far the frame being rendered is from the start of the transition, something along the lines of...

```
Transition = (currrentFrame - startOfTransition)/lengthOfTransition;
```

This means that the host is completely responsible for any user interface for that parameter, either implicit (as in the above editing example) or explicit (with a curve).

Similarly with the 'SourceTime' double parameter in the retimer context. It is up to the host to provide a UI for this, either implicitly (say by stretching a clip's length on the time line) or via an explicit curve. Note that the host is not limited to using a UI that exposes the 'SourceTime' as a curve, alternately it could present a 'speed' parameter, and integrate that to derive a value for 'SourceTime'.

1.9 Thread and Recursion Safety

Hosts are generally multi-threaded, those with a GUI will most likely have an interactive thread and a rendering thread, while any host running on a multi-CPU machine may have a render thread per CPU. Host may batch effects off to a render farm, where the same effect has separate frames rendered on completely different machines. OFX needs to address all these situations.

Threads in the host application can be broken into two categories...

- main threads, where any action may be called
- render threads where only a subset of actions may be called.

For a given effect instance, there can be only one main thread and zero or more render threads. An instance must be able to handle simultaneous actions called on the main and render threads. A plugin can control the number of simultaneous render threads via the *kOfxImageEffectPluginRenderThreadSafety* effect descriptor property.

The only actions that can be called on a render thread are...

- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionEndSequenceRender*
- *kOfxImageEffectActionIsIdentity*
- *kOfxImageEffectActionGetFramesNeeded*
- *kOfxImageEffectActionGetRegionOfDefinition*
- *kOfxImageEffectActionGetRegionsOfInterest*

If a plugin cannot support this multi-threading behaviour, it will need to perform explicit locking itself, using the locking mechanisms in the suites defined in *ofxMultiThread.h*.

This will also mean that the host may need to perform locking on the various function calls over the API. For example, a main and render thread may both simultaneously attempt to access a parameter from a single effect instance. The locking should...

- block write/read access
- not block on read/read access
- be fine grained at the level of individual function calls,
- be transparent to the plugin, so it will block until the call succeeds.

For example, a render thread will only cause a parameter to lock out writes only for the duration of the call that reads the parameter, not for the duration of the whole render action. This will allow a main thread to continue writing to the parameter during a render. This is especially important if you have a custom interactive GUI that you want to keep working during a render call.

Note that a main thread should generally issue an abort to any linked render thread when a parameter or other value affecting the effect (eg: time) has been changed by the user. A re-render should then be issued so that a correct frame is created.

How an effect handles simultaneous calls to render is dealt with in the multi-thread rendering section.

Many hosts get around the problem of sharing a single instance in a UI thread and a render thread by having two instances, one for the user to interact with and a render only one that shadows the UI instance.

1.9.1 Recursive Actions

When running on a main thread, some actions may end up being called recursively. A plug-in must be able to deal with this. For example consider the following sequence of events in a plugin...

1. user sets parameter A in a GUI
2. host issues *kOfxActionInstanceChanged* action
3. plugin traps that and sets parameter B
 1. host issues a new *kOfxActionInstanceChanged* action for parameter B
 2. plugin traps that and changes some internal private state and requests the overlay redraw itself
 1. *kOfxInteractActionDraw* issued to the effect's overlay
 2. plugin draws overlay
 3. *kOfxInteractActionDraw* returns
 3. *kOfxActionInstanceChanged* action for parameter B returns
4. *kOfxActionInstanceChanged* action returns

The image effect actions which may trigger a recursive action call on a single instance are...

- *kOfxActionBeginInstanceChanged*
- *kOfxActionInstanceChanged*
- *kOfxActionEndInstanceChanged*
- *kOfxActionSyncPrivateData*

The interact actions which may trigger a recursive action to be called on the associated plugin instance are...

- *kOfxInteractActionGainFocus*
- *kOfxInteractActionKeyDown*
- *kOfxInteractActionKeyRepeat*
- *kOfxInteractActionKeyUp*
- *kOfxInteractActionLoseFocus*
- *kOfxInteractActionPenDown*
- *kOfxInteractActionPenMotion*
- *kOfxInteractActionPenUp*

The image effect actions which may be called recursively are...

- *kOfxActionBeginInstanceChanged*
- *kOfxActionInstanceChanged*
- *kOfxActionEndInstanceChanged*
- *kOfxImageEffectActionGetClipPreferences*
- *kOfxImageEffectActionGetRegionOfDefinition* (as a result of calling *OfxImageEffectSuiteV1::clipGetImage()* from *kOfxActionInstanceChanged*)
- *kOfxImageEffectActionGetRegionsOfInterest* (as a result of calling *OfxImageEffectSuiteV1::clipGetImage()* from *kOfxActionInstanceChanged*)

The interact actions which may be called recursively are...

- *kOfxInteractActionDraw*

1.10 Coordinate Systems

1.10.1 Spatial Coordinates

All OFX spatial coordinate systems have the positive Y axis pointing up, and the positive X axis pointing right.

As stated above, images are simply some rectangle in a potentially infinite plane of pixels. However, this is an idealisation of what really goes on, as images composed of real pixels have to take into account pixel aspect ratios and proxy render scales, as such they will *not* be in the same space as the image plane. To deal with this, OFX has three spatial coordinate systems

- The Canonical Coordinate System which describes the idealised image plane
- The Pixel Coordinate System which describes coordinates in addressable pixels
- The Normalised Canonical Coordinate System which allows for resolution independent description of parameters

Canonical Coordinates

The idealised image plane is always in a coordinate system of square unscaled pixels. For example a PAL D1 frame occupies (0,0) to (768,576). We call this the *Canonical Coordinate System*.

Many operations take place in canonical coordinates, parameter values are expressed in them while the and RoD and RoI actions report their values back in them.

The Canonical coordinate system is always referenced by double floating point values, generally via a *OfxRectD* structure:

struct **OfxRectD**

Defines two dimensional double region.

Regions are $x1 \leq x < x2$

Infinite regions are flagged by setting

- $x1 = kOfxFlagInfiniteMin$
- $y1 = kOfxFlagInfiniteMin$
- $x2 = kOfxFlagInfiniteMax$
- $y2 = kOfxFlagInfiniteMax$

Public Members

double **x1**

double **y1**

double **x2**

double **y2**

Pixel Coordinates

Real images, where we have to deal with addressable pixels in memory, are in a coordinate system of non-square proxy scaled integer values. So a PAL D1 image, being rendered as a half resolution proxy would be (0,0) to (360, 288), which takes into account both the pixel aspect ratio of 1.067 and a scale factor of 0.5f. We call this the **Pixel Coordinate System**.

The Pixel coordinate system is always referenced by integer values, generally via a `OfxRectI` structure. It is used when referring to operations on actual pixels, and so is how the bounds of images are described and the render window passed to the render action.

Mapping Between The Spatial Coordinate Systems

To map between the two the pixel aspect ratio and the render scale need to be known, and it is a simple case of multiplication and rounding. More specifically, given...

- pixel aspect ratio, PAR , found on the image property `kOfxImagePropPixelAspectRatio`
- render scale in X SX , found on the first dimension of the effect property `kOfxImageEffectPropRenderScale`
- render scale in Y SY , found on the second dimension of the effect property `kOfxImageEffectPropRenderScale`
- field scale in Y FS , this is
 - 0.5 if the image property `kOfxImagePropField` is `kOfxImageFieldLower` or `kOfxImageFieldUpper`
 - 1.0 otherwise.

To map an X and Y coordinates from Pixel coordinates to Canonical coordinates, we perform the following multiplications...

$$\begin{aligned} X' &= (X * PAR) / SX \\ Y' &= Y / (SY * FS) \end{aligned}$$

To map an X and Y coordinates from Canonical coordinates to Pixel coordinates, we perform the following multiplications...

$$\begin{aligned} X' &= (X * SX) / PAR \\ Y' &= Y * SY * FS \end{aligned}$$

The Normalized Coordinate System

Note, normalised parameters and the normalised coordinate system are being deprecated in favour of spatial parameters which can handle the project rescaling without the problems of converting to/from normalised coordinates.

On most editing and compositing systems projects can be moved on resolutions, for example a project may be set up at high definition then have several versions rendered out at different sizes, say a PAL SD version, an NTSC SD version and an HD 720p version.

This causes problems with parameters that describe spatial coordinates. If they are expressed as absolute positions, the values will be incorrect as the project is moved from resolution to resolution. For example, a circle drawn at (384,288) in PAL SD canonical coordinates will be in the centre of the output. Re-render that at 2K film, it will be in the bottom left hand corner, which is probably not the correct spot.

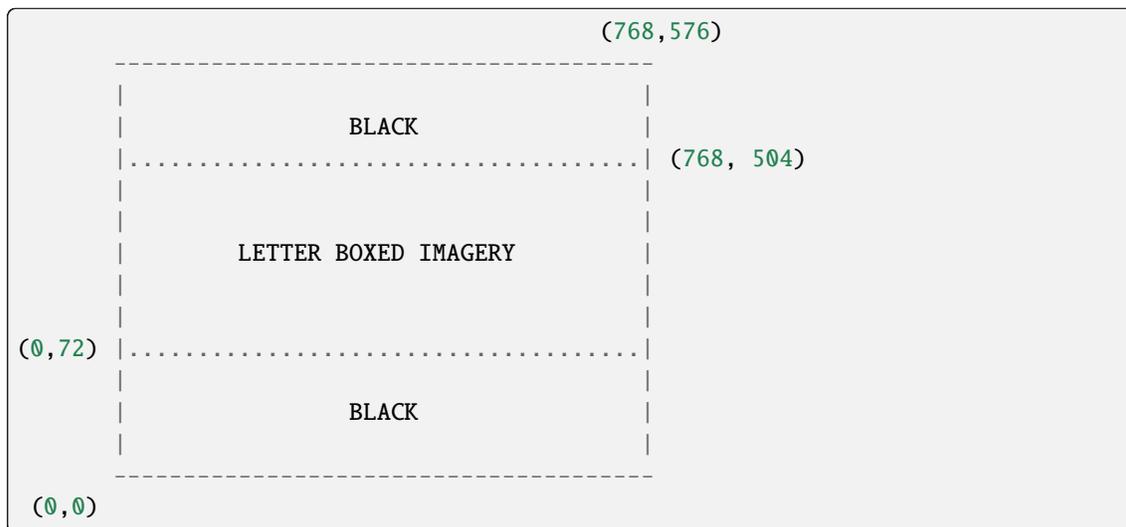
To get around this, OFX allows parameters to be flagged as *normalised*, which is a resolution independent method of representing spatial coordinates. In this coordinate system, a point expressed as (0.5, 0.5) will appear in the centre of the screen, always.

To transform between normalised and canonical coordinates a simple linear equation is required. What that is requires a certain degree of explanation. It involves three two dimensional values...

- the project extent the resolution of the project, eg: PAL SD
- the project size how much of that is used by imagery, eg: the letter box area in a 16:9 PAL SD project
- the project offset the bottom left corner of the extent being used, eg: the BL corner of a 16:9 PAL SD project

As described above, the project extent is the section of the image plane that is covered by an image that is the desired output of the project, so for a PAL SD project you get an extent of 0,0 to 768,576. As the project is always rooted at the origin, so the extent is actually a size.

Project sizes and offsets are a bit less obvious. Consider a project that is going to be output as PAL D1 imagery, the extent will be 0,0 to 768,576. However our example is a letter box 16:9 project, which leaves a strip of black at bottom and top. The size of the letter box is 768 by 432, while the bottom left of the letter box is offset from the origin by 0,77. The ASCII art below shows the details....



So in this example...

- the extent of the project is the full size of the output image, which is 768x576,
- the size of the project is the size of the letter box section, which is 768x432,
- the offset of the project is the bottom left corner of the project window, which is 0,72.

The properties on an effect instance handle allow you to fetch these values...

- *kOfxImageEffectPropProjectExtent* for the extent of the current project,
- *kOfxImageEffectPropProjectSize* for the size of the current project,
- *kOfxImageEffectPropProjectOffset* for the offset of the current project.

So to map from normalised coordinates to canonical coordinates, you use the project size and offset...

- for values that represent a size simply multiply the normalised coordinate by the project size
- for values that represent an absolute position, multiply the normalised coordinate by the project size then add the project origin

To flag to the host that a parameter as normalised, we use the *kOfxParamPropDoubleType* property. Parameters that are so flagged have values set and retrieved by an effect in normalized coordinates. However a host can choose to represent them to the user in whatever space it chooses. The values that this property can take are...

- **kOfxParamDoubleTypeX**

value for the *kOfxParamPropDoubleType* property, indicating a size in canonical coords in the X dimension. See *kOfxParamPropDoubleType*.

A size in the X dimension dimension (1D only), new for 1.2
- **kOfxParamDoubleTypeXAbsolute**

value for the *kOfxParamPropDoubleType* property, indicating an absolute position in canonical coords in the X dimension. See *kOfxParamPropDoubleType*.

A position in the X dimension (1D only), new for 1.2
- **kOfxParamDoubleTypeY**

value for the *kOfxParamPropDoubleType* property, indicating a size in canonical coords in the Y dimension. See *kOfxParamPropDoubleType*.

A size in the Y dimension dimension (1D only), new for 1.2
- **kOfxParamDoubleTypeYAbsolute**

value for the *kOfxParamPropDoubleType* property, indicating an absolute position in canonical coords in the Y dimension. See *kOfxParamPropDoubleType*.

A position in the X dimension (1D only), new for 1.2
- **kOfxParamDoubleTypeXY**

value for the *kOfxParamPropDoubleType* property, indicating a 2D size in canonical coords. See *kOfxParamPropDoubleType*.

A size in the X and Y dimension (2D only), new for 1.2
- **kOfxParamDoubleTypeXYAbsolute**

value for the *kOfxParamPropDoubleType* property, indicating a 2D position in canonical coords. See *kOfxParamPropDoubleType*.

A position in the X and Y dimension (2D only), new for 1.2
- **kOfxParamDoubleTypeNormalisedX**

value for the *kOfxParamPropDoubleType* property, indicating a size normalised to the X dimension. See *kOfxParamPropDoubleType*. [ofxParam.h](#)

Deprecated:

 - V1.3: Deprecated in favour of `::OfxParamDoubleTypeX` V1.4: Removed

Normalised size with respect to the project's X dimension (1D only), deprecated for 1.2
- **kOfxParamDoubleTypeNormalisedXAbsolute**

value for the *kOfxParamPropDoubleType* property, indicating an absolute position normalised to the X dimension. See *kOfxParamPropDoubleType*. [ofxParam.h](#)

Deprecated:

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeXAbsolute` V1.4: Removed

Normalised absolute position on the X axis (1D only), deprecated for 1.2

- **`kOfxParamDoubleTypeNormalisedY`**

value for the `kOfxParamPropDoubleType` property, indicating a size normalised to the Y dimension. See `kOfxParamPropDoubleType`. [#8212; ofxParam.h](#)

Deprecated:

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeY` V1.4: Removed

Normalised size wrt to the project's Y dimension (1D only), deprecated for 1.2

- **`kOfxParamDoubleTypeNormalisedYAbsolute`**

value for the `kOfxParamPropDoubleType` property, indicating an absolute position normalised to the Y dimension. See `kOfxParamPropDoubleType`. [#8212; ofxParam.h](#)

Deprecated:

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeYAbsolute` V1.4: Removed

Normalised absolute position on the Y axis (1D only), deprecated for 1.2

- **`kOfxParamDoubleTypeNormalisedXY`**

value for the `kOfxParamPropDoubleType` property, indicating normalisation to the X and Y dimension for 2D params. See `kOfxParamPropDoubleType`. [#8212; ofxParam.h](#)

Deprecated:

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeXY` V1.4: Removed

Normalised to the project's X and Y size (2D only), deprecated for 1.2

- **`kOfxParamDoubleTypeNormalisedXYAbsolute`**

value for the `kOfxParamPropDoubleType` property, indicating normalisation to the X and Y dimension for a 2D param that can be interpreted as an absolute spatial position. See `kOfxParamPropDoubleType`. [#8212; ofxParam.h](#)

Deprecated:

- V1.3: Deprecated in favour of `kOfxParamDoubleTypeXYAbsolute` V1.4: Removed

Normalised to the projects X and Y size, and is an absolute position on the image plane, deprecated for 1.2.

For example, we have an effect that draws a circle. It has two parameters a 1D double radius parameter and a 2D double position parameter. It would flag the radius to be `kOfxParamDoubleTypeNormalisedX`, fetch the value and scale that by the project size before we render the circle. The host should present such normalised parameters to the user in a *sensible* range. So for a PAL project, it would be from 0..768, where the plug-in sees 0..1.

The position can be handled by the `kOfxParamDoubleTypeNormalisedXYAbsolute` case. In which case the plugin must scale the parameter's value by the project size and add in the project offset. This will allow the positional parameter to be moved between projects transparently.

1.10.2 Temporal Coordinates

Within OFX Image Effects, there is only one temporal coordinate system, this is in output frames referenced to the start of the effect (so the first affected frame = 0). All times within the API are in that coordinate system.

All clip instances have a property that indicates the frames for which they can generate image data. This is *kOfxImageEffectPropFrameRange*, a 2D double property, with the first dimension being the first, and the second being last the time at which the clip will generate data.

Consider the example below, it is showing an effect of 10 frames duration applied to a clip lasting 20 frames. The first frame of the effect is in fact the 5th frame of the clip. Both the input and output have the same frame rate.

Effect					0	1	2	3	4	5	6	7	8	9						
Source	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

In this example, if the effect asks for the source image at time '4', the host will actually return the 9th image of that clip. When queried the output and source clip instances would report the following...

	range[0]	range[1]	FPS
Output	0	9	25
Source	-4	15	25

Consider the slightly more complex example below, where the output has a frame rate twice the input's

Effect										
Source	0	1	2	3	4	5	6	7	8	9

When queried the output and source clips would report the following.

	range[0]	range[1]	FPS
Output	0	9	50
Source	-2	12	25

Using simple arithmetic, any effect that needs to access a specific frame of an input, can do so with the formula...

$$f' = (f - \text{range}[0]) * \text{srcFPS}/\text{outFPS}$$

1.11 Images and Clips

1.11.1 What Is An Image?

Image Effects process images (funny that), this chapter describes images and clips of images, how they behave and how to deal with them.

Firstly some definitions...

- an image is a rectangular array of addressable pixels,
- a clip is a contiguous sequence of images that vary over time.

Images and clips contain pixels, these pixels can currently be of the following types...

- a colour pixel with red, green, blue, alpha components
- a colour pixel with red, green and blue components
- single component 'alpha' images

The components of the pixels can be of the following types...

- 8 bit unsigned byte, with the nominal black and white points at 0 and 255 respectively,
- 16 bit unsigned short, with the nominal black and white points at 0 and 65535 respectively,
- 32 bit float, with the nominal black and white points at 0.0f and 1.0f respectively, component values are not clipped to 0.0f and 1.0f.

Components are packed per pixel in the following manner...

- RGBA pixels as R, G, B, A
- RGB pixels as R, G, B

There are several structs for pixel types in *ofxCore.h* <<https://github.com/ofxa/openfx/blob/master/include/ofxCore.h>> that can be used for raw pixels in OFX.

Images are always left to right, bottom to top, with the pixel data pointer being at the bottom left of the image. The pixels in a scan line are contiguously packed.

Scanlines need *not* be contiguously packed. The number of *bytes* between between a pixel in the same column, but separated by a scan line is known as the *rowbytes* of an image. Rowbytes can be negative, allowing for compositing systems with a native top to bottom scanline order to trivially support bottom to top images.

Clips and images also have a *pixel aspect ratio*, this is how much an actual addressable pixel must be stretched by in X to be square. For example PAL SD images have a pixel aspect ratio of 1.06666.

Images are rectangular, whose integral bounds are in *Pixel coordinates*, with the image being $X1 \leq X < X2$ and $Y1 \leq Y < Y2$, ie: exclusive on the top and right. The bounds represent the amount of data present in the image, which may be larger, smaller or equal to the Region of Definition of the image, depending on the architecture supported by the plugin. The *kOfxImagePropBounds* property on an image holds this information.

An image also contains its RoD in image coordinates, in the *kOfxImagePropRegionOfDefinition* property. The RoD is the maximum area that an image may have pixels in, the bounds are the actual addressable pixels present in an image. This allows for tiled rendering and so on.

Clips have a frame rate, which is the number of frames per second they are to be displayed at. Some clips may be continuously samplable (for example, if they are connected to animating geometry that can be rendered at arbitrary times), if this is so, the frame rate for these clips is set to 0.

Images may be composed of full frames, two fields or a single field, depending on its source and how the effect requests the image be processed. Clips are either full frame sequences or fielded sequences.

Images and clips also have a premultiplication state, this represents how the alpha component and the RGB/YUV components may have interacted.

1.11.2 Defining Clips

During an the effect's describe in context action an effect *must* define the clips mandated for that context, it can also define extra clips that it may need for that context. It does this using the `:cpp:func`OfxImageEffectSuiteV1::clipDefine`` function, the property handle returned by this function is purely for definition purposes only. It has not persistence outside the describe in context action and is distinct to the clip property handles used by instances. The *name* parameter is how you can later access that clip in a plugin instance via the `:cpp:func`OfxImageEffectSuiteV1::clipGetHandle`` function.

During the describe in context action, the plugin sets properties on a clip to control its use. The properties that can be set during a describe in context call are...

- *kOfxPropLabel* to give a user readable name to the clip (the host need not use this, for example in a transition it is redundant),

- *kOfxImageEffectPropSupportedComponents* to specify which components it is willing to accept on that clip,
- *kOfxImageClipPropOptional* to specify if the clip is optional,
- *kOfxImageClipPropFieldExtraction*
specifies how to extract fielded images from a clip, see [this section](#) for more details on field and field rendering
- *kOfxImageEffectPropTemporalClipAccess* whether the effect wants to access images from the clip at times other than the frame being rendered.

Plugins *must* indicate which pixel depths they can process by setting the *kOfxImageEffectPropSupportedPixelDepths* on the plugin handle during the describe action.

Pixel Aspect Ratios, frame rates, fielding, components and pixel depths are constant for the duration of a clip, they cannot be changed from frame to frame.

Note:

- it is an error not to set the *kOfxImageEffectPropSupportedPixelDepths* plugin property during its describe action
 - it is an error not to define a mandated input clip during the describe in context action
 - it is an error not to set the *kOfxImageEffectPropSupportedComponents* on an input clip during describe in context
-

1.11.3 Getting Images From Clips

Clips in instances are retrieved via the `:cpp:func`OfxImageEffectSuiteV1::clipGetHandle`` function. This returns a property handle for the clip in a specific instance. This handle is valid for the duration of the instance.

Images are fetched from a clip via the `OfxImageEffectSuiteV1::clipGetImage()` function. This takes a time and an optional region to extract an image from a given clip. This returns, in a property handle, an image fetched from the clip at a specific time. The handle contains all the information relevant to dealing with that image.

Once fetched, an image must be released via the `OfxImageEffectSuiteV1::clipReleaseImage()` function. All images must be released within the action they were fetched in. You cannot retain an image after an action has returned.

Images may be fetched from an attached clip in the following situations...

- in the *kOfxImageEffectActionRender* action
- in the *kOfxActionInstanceChanged*

and *kOfxActionEndInstanceChanged* actions with a *kOfxPropChangeReason* of *kOfxChangeUserEdited*

A host may not be able to support random temporal access, it flags its ability to do so via the *kOfxImageEffectPropTemporalClipAccess* property. A plugin that wishes to perform random temporal access must set a property of that name on the plugin handle and the clip it wishes to perform random access from.

Note:

- it is an error for a plugin to attempt random temporal image access if the host does not support it
- it is an error for a plugin to attempt random temporal image access

if it has not flagged that it wishes to do so and the clip it wishes to do so from.

1.11.4 Premultiplication And Alpha

All images and clips have a premultiplication state. This is used to indicate how the image should interpret RGB (or YUV) pixels, with respect to alpha. The premultiplication state can be...

kOfxImageOpaque

Used to flag the alpha of an image as opaque

The image is opaque and so has no premultiplication state, but the alpha component in all pixels is set to the white point

kOfxImagePreMultiplied

Used to flag an image as premultiplied

The image is premultiplied by it's alpha

kOfxImageUnPreMultiplied

Used to flag an image as unpremultiplied

The image is unpremultiplied.

This document won't go into the details of premultiplication, but will simply state that OFX takes notice of it and flags images and clips accordingly.

The premultiplication state of a clip is constant over the entire duration of that clip.

1.11.5 Clips and Pixel Aspect Ratios

All clips and images have a pixel aspect ratio, this is how much a 'real' pixel must be stretched by in X to be square. For example PAL D1 images have a pixel aspect ratio of 1.06666.

The property *kOfxImageEffectPropSupportsMultipleClipPARs* is used to control how a plugin deals with pixel aspect ratios. This is both a host and plugin property. For a host it can be set to...

- 0 - the host only supports a single pixel aspect ratio for all clips, input or output, to an effect,
- 1 - the host can support differing pixel aspect ratios for inputs and outputs

For a plugin it can be set to...

- 0 - the plugin expects all pixel aspect ratios to be the same on all clips, input or output
- 1 - the plugin will accept clips of differing pixel aspect ratio.

If a plugin does not accept clips of differing PARs, then the host must resample all images fed to that effect to agree with the output's PAR.

If a plugin does accept clips of differing PARs, it will need to specify the output clip's PAR in the *kOfxImageEffectActionGetClipPreferences* action.

1.11.6 Allocating Your Own Images

Under OFX, the images you fetch from the host have already had their memory allocated. If a plug-in needs to define its own temporary images buffers during processing, or to cache images between actions, then the plug-in should use the image memory allocation routines declared in *OfxImageEffectSuiteV1*. The reason for this is that many host have special purpose memory pools they manage to optimise memory usage as images can chew up memory very rapidly (eg: a 2K RGBA floating point film plate is 48 MBytes).

For general purpose (as in less than a megabyte) memory allocation, you should use the memory suite in *ofxMemory.h*. OFX provides four functions to deal with image memory. These are,

- *OfxImageEffectSuiteV1::imageMemoryAlloc()*
- *OfxImageEffectSuiteV1::imageMemoryFree()*
- *OfxImageEffectSuiteV1::imageMemoryLock()*
- *OfxImageEffectSuiteV1::imageMemoryUnlock()*

A host needs to be able defragment its image memory pool, potentially moving the contents of the memory you have allocated to another address, even saving it to disk under its own virtual memory caching scheme. Because of this when you request a block of memory, you are actually returned a handle to the memory, not the memory itself. To use the memory you must first lock the memory via the *imageMemoryLock* call, which will then return a pointer to the locked block of memory.

During a single action, there is generally no need to lock/unlock any temporary buffers you may have allocated via this mechanism. However image memory that is cached between actions should always be unlocked while it is not actually being used. This allows a host to do what it needs to do to optimise memory usage.

Note that locks and unlocks nest. This implies that there is a lock count kept on the memory handle, also not that this lock count cannot be negative. So unlocking a completely unlocked handle has no effect.

An example is below...

```
// get a memory handle
OfxImageMemoryHandle memHandle;
gEffectSuite->imageMemoryAlloc(0, imageSize, &memHandle);

// lock the handle and get a pointer
void *memPtr;
gEffectSuite->imageMemoryLock(memHandle, &memPtr);

... // do stuff with our pointer

// now unlock it
gEffectSuite->imageMemoryUnlock(memHandle);

// lock it again, note that this may give a completely different address to
↳the last lock
gEffectSuite->imageMemoryLock(memHandle, &memPtr);

... // do more stuff

// unlock it again
gEffectSuite->imageMemoryUnlock(memHandle);
```

(continues on next page)

(continued from previous page)

```
// delete it all
gEffectSuite->imageMemoryFree(memHandle);
```

1.12 Effect Parameters

1.12.1 Introduction

Nearly all plug-ins have some sort of parameters that control their behaviour, the radius of a circle drawer, the frequencies to filter out of an audio signal, the colour of a lens flare and so on.

Seeing as hosts already provide for the general management of their own native parameters (eg: persistence, interface, animation etc...), it would make no sense to force plug-ins to do this all themselves.

The OFX Parameters Suite is the means by which parameters are defined and used by the plug-in but maintained by the host. It is defined in the `ofxParam.h` header file.

Note that the entire state of the plugin is encoded in the value of its parameter set. If you need to persist some sort of private data, you must do so by setting param values in the effects. The `kOfxActionSyncPrivateData` is an action that tells you when to flush any values that need persisting out to the effects param set. You can reconstruct your private data during the `kOfxActionCreateInstance`.

1.12.2 Defining Parameters

A plugin needs to define its parameters during a describe action. It does this with the `OfxParameterSuiteV1::paramDefine()` function, which returns a handle to a parameter *description*. Parameters cannot currently be defined outside of the plugins describe actions.

Parameters are uniquely labelled within a plugin with an ASCII null terminated C-string. This name is not necessarily meant to be end-user readable, various properties are provided to set the user visible labels on the param.

All parameters hold properties, though the exact set of properties on a parameter is dependent on the type of the parameter.

A parameter's handle comes in two slightly different flavours. The handle returned inside a plugin's describe action is not an actual instance of a parameter, it is there for the purpose description only. You can only set properties on that handle (eg: label, min/max value, default...), you cannot get values from it or set values in it. The parameters defined in the describe action will common to all instances of a plugin.

The handle returned by `OfxParameterSuiteV1::paramGetHandle()` outside of a describe action will be a working instance of a parameter, you can still set (some) properties of the parameter, and all the get/set value functions are now usable.

1.12.3 Getting and Setting Parameter Values

During rendering and interactions, a plugin gets its param values via `OfxParameterSuiteV1::paramGetValue()` or `OfxParameterSuiteV1::paramGetValueAtTime()`.

To set param values, a plugin can use `OfxParameterSuiteV1::paramSetValue()` or `OfxParameterSuiteV1::paramSetValueAtTime()`.

In addition, `OfxParameterSuiteV1` has functions for manipulating keyframes, copying and editing params, getting their properties, and getting derivatives and integrals of param values.

1.12.4 Parameter Types

There are eighteen types of parameter. These are

- **kOfxParamTypeInteger**
String to identify a param as a single valued integer.
 - **kOfxParamTypeInteger2D**
String to identify a param as a Two dimensional integer point parameter.
 - **kOfxParamTypeInteger3D**
String to identify a param as a Three dimensional integer parameter.
 - **kOfxParamTypeDouble**
String to identify a param as a Single valued floating point parameter
 - **kOfxParamTypeDouble2D**
String to identify a param as a Two dimensional floating point parameter.
 - **kOfxParamTypeDouble3D**
String to identify a param as a Three dimensional floating point parameter.
 - **kOfxParamTypeRGB**
String to identify a param as a Red, Green and Blue colour parameter.
 - **kOfxParamTypeRGBA**
String to identify a param as a Red, Green, Blue and Alpha colour parameter.
 - **kOfxParamTypeBoolean**
String to identify a param as a Single valued boolean parameter.
 - **kOfxParamTypeChoice**
String to identify a param as a Single valued, ‘one-of-many’ parameter.
 - **kOfxParamTypeStrChoice**
String to identify a param as a string-valued ‘one-of-many’ parameter.
- Since**
Version 1.5
- **kOfxParamTypeString**
String to identify a param as a String (UTF8) parameter.
 - **kOfxParamTypeCustom**
String to identify a param as a Plug-in defined parameter.

- **kOfxParamTypePushButton**
String to identify a param as a PushButton parameter.
- **kOfxParamTypeGroup**
String to identify a param as a Grouping parameter.
- **kOfxParamTypePage**
String to identify a param as a page parameter.
- **kOfxParamTypeParametric**
String to identify a param as a single valued integer.

1.12.5 Multidimensional Parameters

Some parameter types are multi dimensional, these are...

- *kOfxParamTypeDouble2D*
- *kOfxParamTypeInteger2D*
- *kOfxParamTypeDouble3D*
- *kOfxParamTypeInteger3D*
- *kOfxParamTypeRGB*
- *kOfxParamTypeRGBA*
- *kOfxParamTypeParametric*

These parameters are treated in an atomic manner, so that all dimensions are set/retrieved simultaneously. This applies to keyframes as well.

The non colour parameters have an implicit 'X', 'Y' and 'Z' dimension, and any interface should display them with such labels.

1.12.6 Integer Parameters

These are typed by *kOfxParamTypeInteger*, *kOfxParamTypeInteger2D* and *kOfxParamTypeInteger3D*.

Integer parameters are of 1, 2 and 3D varieties and contain integer values, between INT_MIN and INT_MAX.

1.12.7 Double Parameters

These are typed by *kOfxParamTypeDouble*, *kOfxParamTypeDouble2D* and *kOfxParamTypeDouble3D*.

Double parameters are of 1, 2 and 3D varieties and contain double precision floating point values.

1.12.8 Colour Parameters

These are typed by *kOfxParamTypeRGB* and *kOfxParamTypeRGBA*.

Colour parameters are 3 or 4 dimensional double precision floating point parameters. They are displayed using the host's appropriate interface for a colour. Values are always normalised in the range [0 .. 1], with 0 being the nominal black point and 1 being the white point.

1.12.9 Boolean Parameters

This is typed by *kOfxParamTypeBoolean*.

Boolean parameters are integer values that can have only one of two values, 0 or 1.

1.12.10 Choice Parameters

This is typed by *kOfxParamTypeChoice*.

Choice parameters are integer values from 0 to N-1, which correspond to N labeled options, but see *kOfxParamPropChoiceOrder* and the section below this for how to change that.

Choice parameters have their individual options set via the *kOfxParamPropChoiceOption* property, for example

```
gPropHost->propSetString(myChoiceParam, kOfxParamPropChoiceOption, 0, "1st_
↳Choice");
gPropHost->propSetString(myChoiceParam, kOfxParamPropChoiceOption, 1, "2nd_
↳Choice");
gPropHost->propSetString(myChoiceParam, kOfxParamPropChoiceOption, 2, "3rd_
↳Choice");
...
gPropHost->propSetString(myChoiceParam, kOfxParamPropChoiceOption, n, "nth_
↳Choice");
```

It is an error to have gaps in the choices after the describe action has returned.

Note: plugins can change the *text* of options strings in new versions with no compatibility impact, since the host should only store the index. But they should not change the *order* of options without using *kOfxParamPropChoiceOrder*.

If no default value is set by the plugin, the host should use the first defined option (index 0).

Setting Choice Param Order

As of OFX v1.5, plugins can optionally specify the order in which the host should display each choice option, using *kOfxParamPropChoiceOrder*.

This property contains a set of integers, of the same length as the options for the choice parameter. If the host supports this property, it should sort the options and the order list together, and display the options in increasing order.

This property is useful when changing order of choice param options, or adding new options in the middle, in a new version of the plugin.

```
// Plugin v1:
Option = {"OptA", "OptB", "OptC"}
Order  = {0, 1, 2} // default, or explicit
```

(continues on next page)

(continued from previous page)

```
// Plugin v2:
// add NewOpt at the end of the list, but specify order so it comes one before
↳the end in the UI
// Will display OptA / OptB / NewOpt / OptC
Option = {"OptA", "OptB", "OptC", "NewOpt"}
Order  = {0, 1, 3, 2} // or anything that sorts the same, e.g. {1, 100, 300,
↳200}
```

In this case if the user had selected “OptC” in v1, and then loaded the project in v2, “OptC” will still be selected even though it is now the 4th option, and the plugin will get the param value 2, as it did in its previous version.

The default, if unspecified, is ordinal integers starting from zero, so the options are displayed in their natural order.

Values may be arbitrary 32-bit integers. The same value must not occur more than once in the order list; behavior is undefined if the same value occurs twice in the list. Plugins should use non-negative values; some hosts may choose to hide options with negative Order values.

Note that *kOfxParamPropChoiceOrder* does not affect project storage or operation; it is only used by the host UI. This way it is 100% backward compatible; even if the plugin sets it and the host doesn’t support it, the plugin will still work as usual. Its options will just appear with the new ones at the end rather than the preferred order.

To query whether a host supports this, a plugin should attempt to set the property and check the return status. If the host does not support *kOfxParamPropChoiceOrder*, a plugin should not insert new values into the middle of the options list, nor reorder the options, in a new version, otherwise old projects will not load properly.

Note: this property does not help if a plugin wants to *remove* an option. One way to handle that case is to define a new choice param in v2 and hide the old v1 param, then use some custom logic to populate the v2 param appropriately.

Also in 1.5, see the new *kOfxParamTypeStrChoice* param type for another way to do this: the plugin specifies a set of string values as well as user-visible options, and the host stores the string value. Plugins can then change the UI order at will in new versions, by reordering the options and enum arrays.

Available since 1.5.

1.12.11 String-Valued Choice Parameters

This is typed by *kOfxParamTypeStrChoice*.

String Choice (“StrChoice”) parameters are string-valued, unlike standard Choice parameters. This way plugins may change the text and order of the choices in new versions as desired, without sacrificing compatibility. The host stores the string value rather than the index of the option, and when loading a project, finds and selects the option with the corresponding enum.

Choice parameters have their individual options and enums set via the *kOfxParamPropChoiceOption* and *kOfxParamPropChoiceEnum* properties, for example

```
gPropHost->propSetString(myChoiceParam, kOfxParamPropChoiceOption, 0, "1st
↳Choice");
gPropHost->propSetString(myChoiceParam, kOfxParamPropChoiceOption, 1, "2nd
↳Choice");
gPropHost->propSetString(myChoiceParam, kOfxParamPropChoiceOption, 2, "3rd
↳Choice");
...
// enums: string values to be returned as param value, and stored by the host
↳in the project
gPropHost->propSetString(myChoiceParam, kOfxParamPropChoiceEnum, 0, "choice-1
```

(continues on next page)

(continued from previous page)

```

↪");
gPropHost->propSetString(myChoiceParam, kOfxParamPropChoiceEnum, 1, "choice-2
↪");
gPropHost->propSetString(myChoiceParam, kOfxParamPropChoiceEnum, 2, "choice-3
↪");

```

The default value of a `StrChoice` param must be one of the specified enums, or the behavior is undefined. If no default value is set by the plugin, the host should use the first defined option (index 0).

It is an error to have gaps in the choices after the describe action has returned. The `Option` and `Enum` arrays must be of the same length, otherwise the behavior is undefined.

If a plugin removes enums in later versions and a project is saved with the removed enum, behavior is undefined, but it is recommended that the host use the default value in that case.

To check for availability of this param type, a plugin may check the host property `kOfxParamHostPropSupportsStrChoice`.

`StrChoice` parameters may also be reordered using `kOfxParamPropChoiceOrder`; see the previous section.

Available since 1.5.

1.12.12 String Parameters

This is typed by `kOfxParamTypeString`.

String parameters contain null terminated `char *` UTF8 C strings. They can be of several different variants, which is controlled by the `kOfxParamPropStringMode` property, these are

- **kOfxParamStringIsSingleLine**
Used to set a string parameter to be single line, value to be passed to a `kOfxParamPropStringMode` property.
- **kOfxParamStringIsMultiLine**
Used to set a string parameter to be multiple line, value to be passed to a `kOfxParamPropStringMode` property.
- **kOfxParamStringIsFilePath**
Used to set a string parameter to be a file path, value to be passed to a `kOfxParamPropStringMode` property.
- **kOfxParamStringIsDirectoryPath**
Used to set a string parameter to be a directory path, value to be passed to a `kOfxParamPropStringMode` property.
- **kOfxParamStringIsLabel**
Use to set a string parameter to be a simple label, value to be passed to a `kOfxParamPropStringMode` property
- **kOfxParamStringIsRichTextFormat**
String value on the `kOfxParamPropStringMode` property of a string parameter (added in 1.3)

1.12.13 Group Parameters

This is typed by *kOfxParamTypeGroup*.

Group parameters allow all parameters to be arranged in a tree hierarchy. They have no value, they are purely a grouping element.

All parameters have a *kOfxParamPropParent* property, which is a string property naming the group parameter which is its parent.

The empty string "" is used to label the root of the parameter hierarchy, which is the default parent for all parameters.

Parameters inside a group are ordered by their order of addition to that group, which implies parameters in the root group are added in order of definition.

Any host based hierarchical GUI should use this hierarchy to order parameters (eg: animation sheets).

1.12.14 Page Parameters

This is typed by *kOfxParamTypePage*.

Page parameters are covered in detail in their own *section*.

1.12.15 Custom Parameters

This is typed by *kOfxParamTypeCustom*.

Custom parameters contain null terminated char * C strings, and may animate. They are designed to provide plugins with a way of storing data that is too complicated or impossible to store in a set of ordinary parameters.

If a custom parameter animates, it must set its *kOfxParamPropCustomInterpCallbackV1* property, which points to a function with the following signature:

OfxStatus() **OfxCustomParamInterpFuncV1** (**OfxParamSetHandle** instance, **OfxPropertySetHandle** inArgs, **OfxPropertySetHandle** outArgs)

Function prototype for custom parameter interpolation callback functions.

- instance the plugin instance that this parameter occurs in
- inArgs handle holding the following properties...
 - kOfxPropName - the name of the custom parameter to interpolate
 - kOfxPropTime - absolute time the interpolation is occurring at
 - kOfxParamPropCustomValue - string property that gives the value of the two keyframes to interpolate, in this case 2D
 - kOfxParamPropInterpolationTime - 2D double property that gives the time of the two keyframes we are interpolating
 - kOfxParamPropInterpolationAmount - 1D double property indicating how much to interpolate between the two keyframes
- outArgs handle holding the following properties to be set
 - kOfxParamPropCustomValue - the value of the interpolated custom parameter, in this case 1D

This function allows custom parameters to animate by performing interpolation between keys.

The plugin needs to parse the two strings encoding keyframes on either side of the time we need a value for. It should then interpolate a new value for it, encode it into a string and set the *kOfxParamPropCustomValue* property with this on the outArgs handle.

The interp value is a linear interpolation amount, however this may be derived from a cubic (or other) curve.

This function is used to interpolate keyframes in custom params.

Custom parameters have no interface by default. However,

- if they animate, the host's animation sheet/editor should present a keyframe/curve representation to allow positioning of keys and control of interpolation. The 'normal' (ie: paged or hierarchical) interface should not show any gui.
- if the custom param sets its *kOfxParamPropInteractV1* property, this should be used by the host in any normal (ie: paged or hierarchical) interface for the parameter.

Custom parameters are mandatory, as they are simply ASCII C strings. However, animation of custom parameters and support for an in editor interact is optional.

1.12.16 Push Button Parameters

This is typed by *kOfxParamTypePushButton*.

Push button parameters have no value, they are there so a plugin can detect if they have been pressed and perform some action. If pressed, a *kOfxActionInstanceChanged* action will be issued on the parameter with a *kOfxPropChangeReason* of *kOfxChangeUserEdited*.

1.12.17 Animation

By default the following parameter types animate...

- *kOfxParamTypeInteger*
- *kOfxParamTypeInteger2D*
- *kOfxParamTypeInteger3D*
- *kOfxParamTypeDouble*
- *kOfxParamTypeDouble2D*
- *kOfxParamTypeDouble3D*
- *kOfxParamTypeRGBA*
- *kOfxParamTypeRGB*

The following types cannot animate...

- *kOfxParamTypeGroup*
- *kOfxParamTypePage*
- *kOfxParamTypePushButton*

The following may animate, depending on the host. Properties exist on the host to check this. If the host does support animation on them, then they do **not** animate by default. They are...

- *kOfxParamTypeCustom*
- *kOfxParamTypeString*

- *kOfxParamTypeBoolean*
- *kOfxParamTypeChoice*
- *kOfxParamTypeStrChoice*

By default the *OfxParameterSuiteV1::paramGetValue()* will get the ‘current’ value of the parameter. To access values in a potentially animating parameter, use the *OfxParameterSuiteV1::paramGetValueAtTime()* function.

Keys can be manipulated in a parameter using a variety of functions, these are...

- *OfxParameterSuiteV1::paramSetValueAtTime()*
- *OfxParameterSuiteV1::paramGetNumKeys()*
- *OfxParameterSuiteV1::paramGetKeyTime()*
- *OfxParameterSuiteV1::paramGetKeyIndex()*
- *OfxParameterSuiteV1::paramDeleteKey()*
- *OfxParameterSuiteV1::paramDeleteAllKeys()*

1.12.18 Parameter Interfaces

Parameters will be presented to the user in some form of interface. Typically on most host systems, this comes in three varieties...

- a paged layout, with parameters spread over multiple controls pages (eg: the FLAME control pages)
- a hierarchical layout, with parameters presented in a grouped tree (eg: the After Effects ‘effects’ window)
- an animation sheet, showing animation curves and key frames. Typically this is hierarchical.

Most systems have an animation sheet and present one of either the paged or the hierarchical layouts.

Because a hierarchy of controls is explicitly set during plugin definition, the case of the animation sheet and hierarchial GUIs are taken care of explicitly.

1.12.19 Paged Parameter Editors

A paged layout of controls is difficult to standardise, as the size of the page and controls, how the controls are positioned on the page, how many controls appear on a page etc... depend very much upon the host implementation. A paged layout is ideally best described in the .XML resource supplied by the plugin, however a fallback page layout can be specified in OFX via the *kOfxParamTypePage* parameter type.

Several host properties are associated with paged layouts, these are...

- *kOfxParamHostPropMaxPages* The maximum number of pages you may use, 0 implies an unpagged layout
- *kOfxParamHostPropPageRowColumnCount* The number of rows and columns for parameters in the paged layout.

Each page parameter represents a page of controls. The controls in that page are set by the plugin using the *kOfxParamPropPageChild* multi-dimensional string. For example...

```
OfxParamHandle page;
gHost->paramDefine(plugin, kOfxParamTypePage, "Main", &page);

propHost->propSetString(page, kOfxParamPropPageChild, 0, "size"); // add
↪the size parameter to the top left of the page
```

(continues on next page)

(continued from previous page)

```

propHost->propSetString(page, kOfxParamPropPageChild, 1, kOfxParamPageSkipRow);
↪ // skip a row
propHost->propSetString(page, kOfxParamPropPageChild, 2, "centre"); // add
↪ the centre parameter
propHost->propSetString(page, kOfxParamPropPageChild, 3,
↪ kOfxParamPageSkipColumn); // skip a column, we are now at the top of the
↪ next column
propHost->propSetString(page, kOfxParamPropPageChild, 4, "colour"); // add the
↪ colour parameter

```

The host then places the parameters on that page in the order they were added, starting at the top left and going down columns, then across rows as they fill.

Note that there are two pseudo parameter names used to help control layout:

kOfxParamPageSkipRow

Pseudo parameter name used to skip a row in a page layout.

Passed as a value to the *kOfxParamPropPageChild* property.

See *ParametersInterfacesPagedLayouts* for more details.

kOfxParamPageSkipColumn

Pseudo parameter name used to skip a row in a page layout.

Passed as a value to the *kOfxParamPropPageChild* property.

See *ParametersInterfacesPagedLayouts* for more details.

These will help control how parameters are added to a page, allowing vertical or horizontal slots to be skipped.

A host sets the order of pages by using the instance's *kOfxPluginPropParamPageOrder* property. Note that this property can vary from context to context, so you can exclude pages in contexts they are not useful in. For example...

```

OfxStatus describeInContext(OfxImageEffectHandle plugin)
{
...
    // order our pages of controls
    propHost->propSetString(paramSetProp, kOfxPluginPropParamPageOrder, 0,
↪ "Main");
    propHost->propSetString(paramSetProp, kOfxPluginPropParamPageOrder, 1,
↪ "Sampling");
    propHost->propSetString(paramSetProp, kOfxPluginPropParamPageOrder, 2,
↪ "Colour Correction");
    if(isGeneralContext)
        propHost->propSetString(paramSetProp, kOfxPluginPropParamPageOrder, 3,
↪ "Dance! Dance! Dance!");
...
}

```

Note: Parameters can be placed on more than a single page (this is often useful). Group parameters cannot be added to a page. Page parameters cannot be added to a page or group.

1.12.20 Instance changed callback

Whenever a parameter's value changes, the host is expected to issue a call to the *kOfxActionInstanceChanged* action with the name of the parameter that changed and a reason indicating who triggered the change:

kOfxChangeUserEdited

String used as a value to *kOfxPropChangeReason* to indicate a user has changed something.

kOfxChangePluginEdited

String used as a value to *kOfxPropChangeReason* to indicate the plug-in itself has changed something.

kOfxChangeTime

String used as a value to *kOfxPropChangeReason* to a time varying object has changed due to a time change.

1.12.21 Parameter Undo/Redo

Hosts usually retain an undo/redo stack, so users can undo changes they make to a parameter. Often undos and redos are grouped together into an undo/redo block, where multiple parameters are dealt with as a single undo/redo event. Plugins need to be able to deal with this cleanly.

Parameters can be excluded from being undone/redone if they set the *kOfxParamPropCanUndo* property to 0.

If the plugin changes parameters values by calling the get and set value functions, they will ordinarily be put on the undo stack, one event per parameter that is changed. If the plugin wants to group sets of parameter changes into a single undo block and label that block, it should use the *OfxParameterSuiteV1::paramEditBegin()* and *OfxParameterSuiteV1::paramEditEnd()* functions.

An example would be a 'preset' choice parameter in a sky simulation whose job is to set other parameters to values that achieve certain looks, eg "Dusk", "Midday", "Stormy", "Night" etc... This parameter has a value change callback which looks for *kOfxChangeUserEdited* then sets other parameters, sky colour, cloud density, sun position etc... It also resets itself to the first choice, which says "Example Skys...".

Rather than have many undo events appear on the undo stack for each individual parameter change, the effect groups them via the *paramEditBegin/paramEditEnd* and gets a single undo event. The 'preset' parameter would also not want to be undoable as it such an event is redundant. Note that as the 'preset' has been changed it will be sent another instance changed action, however it will have a reason of *kOfxChangePluginEdited*, which it ignores and so stops an infinite loop occurring.

1.12.22 XML Resource Specification for Parameters

Parameters can have various properties overridden via a separate XML based resource file.

1.12.23 Parameter Persistence

All parameters flagged with the *kOfxParamPropPersistent* property will persist when an effect is saved. How the effect is saved is completely up to the host, it may be in a file, a data base, where ever. We call a saved set of parameters a *setup*. A host will need to save the major version number of the plugin, as well as the plugin's unique identifier, in any setup.

When an host loads a set up it should do so in the following manner...

1. examines the setup for the major version number.

2. find a matching plugin with that major version number, if multiple minor versions exist, the plugin with the largest minor version should be used.
3. creates an instance of that plugin with its set of parameters.
4. sets all those parameters to the defaults specified by the plugin.
5. examines the setup for any persistent parameters, then sets the instance's parameters to any found in it.
6. calls create instance on the plugin.

It is *not* an error for a parameter to exist in the plugin but not the setup, and vice versa. This allows a plugin developer to modify parameter sets between point releases, generally by adding new params. The developer should be sure that the default values of any new parameters yield the same behaviour as before they were added, otherwise it would be a breach of the 'major version means compatibility' rule.

1.12.24 Parameter Properties Whose Type Vary

Some properties type depends on the kind of the parameter, eg: *kOfxParamPropDefault* is an int for a integer parameter but a double X 2 for a *kOfxParamTypeDouble2D* parameter.

The variant property types are as follows...

- *kOfxParamTypeInteger*
int X 1
- *kOfxParamTypeDouble*
double X 1
- *kOfxParamTypeBoolean*
int X 1
- *kOfxParamTypeChoice*
int X 1
- *kOfxParamTypeStrChoice*
char * X 1
- *kOfxParamTypeRGBA*
double X 4 (normalised to 0..1 range)
- *kOfxParamTypeRGB*
double X 3 (normalised to 0..1 range)
- *kOfxParamTypeDouble2D*
double X 2
- *kOfxParamTypeInteger2D*
int X 2
- *kOfxParamTypeDouble3D*
double X 3
- *kOfxParamTypeInteger3D*
int X 3
- *kOfxParamTypeString*
char * X 1
- *kOfxParamTypeCustom*
char * X 1

- ***kOfxParamTypePushButton***
none

1.12.25 Types of Double Parameters

Double parameters can be used to represent a variety of data, by flagging what a double parameter is representing, a plug-in allows a host to represent to the user a more appropriate interface than a raw numerical value. Double parameters have the *kOfxParamPropDoubleType* property, which gives some meaning to the value. This can be one of...

- ***kOfxParamDoubleTypePlain***
value for the *kOfxParamPropDoubleType* property, indicating the parameter has no special interpretation and should be interpreted as a raw numeric value.
- ***kOfxParamDoubleTypeAngle***
value for the *kOfxParamDoubleTypeAngle* property, indicating the parameter is to be interpreted as an angle. See *kOfxParamPropDoubleType*.
- ***kOfxParamDoubleTypeScale***
value for the *kOfxParamPropDoubleType* property, indicating the parameter is to be interpreted as a scale factor. See *kOfxParamPropDoubleType*.
- ***kOfxParamDoubleTypeTime***
value for the *kOfxParamDoubleTypeAngle* property, indicating the parameter is to be interpreted as a time. See *kOfxParamPropDoubleType*.
- ***kOfxParamDoubleTypeAbsoluteTime***
value for the *kOfxParamDoubleTypeAngle* property, indicating the parameter is to be interpreted as an absolute time from the start of the effect. See *kOfxParamPropDoubleType*.
- ***kOfxParamDoubleTypeX***
value for the *kOfxParamPropDoubleType* property, indicating a size in canonical coords in the X dimension. See *kOfxParamPropDoubleType*.
- ***kOfxParamDoubleTypeXAbsolute***
value for the *kOfxParamPropDoubleType* property, indicating an absolute position in canonical coords in the X dimension. See *kOfxParamPropDoubleType*.
- ***kOfxParamDoubleTypeY***
value for the *kOfxParamPropDoubleType* property, indicating a size in canonical coords in the Y dimension. See *kOfxParamPropDoubleType*.
- ***kOfxParamDoubleTypeYAbsolute***
value for the *kOfxParamPropDoubleType* property, indicating an absolute position in canonical coords in the Y dimension. See *kOfxParamPropDoubleType*.
- ***kOfxParamDoubleTypeXY***
value for the *kOfxParamPropDoubleType* property, indicating a 2D size in canonical coords. See *kOfxParamPropDoubleType*.

- **kOfxParamDoubleTypeXYAbsolute**
value for the *kOfxParamPropDoubleType* property, indicating a 2D position in canonical coords. See *kOfxParamPropDoubleType*.
- **kOfxParamDoubleTypeNormalisedX**
value for the *kOfxParamPropDoubleType* property, indicating a size normalised to the X dimension. See *kOfxParamPropDoubleType*. [#8212](#); *ofxParam.h*

Deprecated:
– V1.3: Deprecated in favour of `::OfxParamDoubleTypeX` V1.4: Removed
- **kOfxParamDoubleTypeNormalisedXAbsolute**
value for the *kOfxParamPropDoubleType* property, indicating an absolute position normalised to the X dimension. See *kOfxParamPropDoubleType*. [#8212](#); *ofxParam.h*

Deprecated:
– V1.3: Deprecated in favour of `::OfxParamDoubleTypeXAbsolute` V1.4: Removed
- **kOfxParamDoubleTypeNormalisedY**
value for the *kOfxParamPropDoubleType* property, indicating a size normalised to the Y dimension. See *kOfxParamPropDoubleType*. [#8212](#); *ofxParam.h*

Deprecated:
– V1.3: Deprecated in favour of `::OfxParamDoubleTypeY` V1.4: Removed
- **kOfxParamDoubleTypeNormalisedYAbsolute**
value for the *kOfxParamPropDoubleType* property, indicating an absolute position normalised to the Y dimension. See *kOfxParamPropDoubleType*. [#8212](#); *ofxParam.h*

Deprecated:
– V1.3: Deprecated in favour of `::OfxParamDoubleTypeYAbsolute` V1.4: Removed
- **kOfxParamDoubleTypeNormalisedXY**
value for the *kOfxParamPropDoubleType* property, indicating normalisation to the X and Y dimension for 2D params. See *kOfxParamPropDoubleType*. [#8212](#); *ofxParam.h*

Deprecated:
– V1.3: Deprecated in favour of `::OfxParamDoubleTypeXY` V1.4: Removed
- **kOfxParamDoubleTypeNormalisedXYAbsolute**
value for the *kOfxParamPropDoubleType* property, indicating normalisation to the X and Y dimension for a 2D param that can be interpreted as an absolute spatial position. See *kOfxParamPropDoubleType*. [#8212](#); *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of *kOfxParamDoubleTypeXYAbsolute* V1.4: Removed

1.12.26 Plain Double Parameters

Double parameters with their *kOfxParamPropDoubleType* property set to *kOfxParamDoubleTypePlain* are uninterpreted. The values represented to the user are what is reported back to the effect when values are retrieved. 1, 2 and 3D parameters can be flagged as *kOfxParamDoubleTypePlain*, which is the default.

For example a physical simulation plugin might have a ‘mass’ double parameter, which is in kilograms, which should be displayed and used as a raw value.

1.12.27 Angle Double Parameters

Double parameters with their *kOfxParamPropDoubleType* property set to *kOfxParamDoubleTypeAngle* are interpreted as angles. The host could use some fancy angle widget in it’s interface, representing degrees, angles mils whatever. However, the values returned to a plugin are always in degrees. Applicable to 1, 2 and 3D parameters.

For example a plugin that rotates an image in 3D would declare a 3D double parameter and flag that as an angle parameter and use the values as Euler angles for the rotation.

1.12.28 Scale Double Parameters

Double parameters with their *kOfxParamPropDoubleType* property set to *kOfxParamDoubleTypeScale* are interpreted as scale factors. The host can represent these as 1..100 percentages, 0..1 scale factors, fractions or whatever is appropriate for its interface. However, the plugin sees these as a straight scale factor, in the 0..1 range. Applicable to 1, 2 and 3D parameters.

For example a plugin that scales the size of an image would declare a ‘image scale’ parameter and use the raw value of that to scale the image.

1.12.29 Time Double Parameters

Double parameters with their *kOfxParamPropDoubleType* property set to *kOfxParamDoubleTypeTime* are interpreted as a time. The host can represent these as frames, seconds, milliseconds, millennia or whatever it feels is appropriate. However, a visual effect plugin sees such values in ‘frames’. Applicable only to 1D double parameters. It is an error to set this on any other type of double parameter.

For example a plugin that does motion blur would have a ‘shutter time’ parameter and flags that as a time parameter. The value returned would be used as the length of the shutter, in frames.

1.12.30 Absolute Time Double Parameters

Double parameters with their *kOfxParamPropDoubleType* property set to *kOfxParamDoubleTypeAbsoluteTime* are interpreted as an absolute time from the beginning of the effect. The host can represent these as frames, seconds, milliseconds, millennia or whatever it feels is appropriate. However, a plugin sees such values in ‘frames’ from the beginning of a clip. Applicable only to 1D double parameters. It is an error to set this on any other type of double parameter.

For example a plugin that stabilises all the images in a clip to a specific frame would have a *reference frame* parameter and declare that as an absolute time parameter and use its value to fetch a frame to stabilise against.

1.12.31 Spatial Parameters

Parameters that can represent a size or position are essential. To that end there are several values of the `kOfxParamPropDoubleType` that say it should be interpreted as a size or position, in either one or two dimensions.

The original OFX API only specified *normalised* parameters, this proved to be somewhat more of a problem than expected. With the 1.2 version of the API, *spatial* parameters were introduced. Ideally these should be used and the normalised parameter types should be deprecated.

Plugins can check `kOfxPropAPIVersion` to see if these new parameter types are supported, in hosts with version 1.2 or greater they will be.

See the section on coordinate systems to understand some of the terms being discussed.

Spatial Double Parameters

These parameter types represent a size or position in one or two dimensions in *Canonical Coordinate*. The host and plug-in get and set values in this coordinate system. Scaling to *Pixel Coordinate* is the responsibility of the effect.

The default value of a spatial parameter can be set in either a normalised coordinate system or the canonical coordinate system. This is controlled by the `kOfxParamPropDefaultCoordinateSystem` on the parameter descriptor with one of the following value:

kOfxParamCoordinatesCanonical

Define the canonical coordinate system.

kOfxParamCoordinatesNormalised

Define the normalised coordinate system.

Parameters can choose to be spatial in several ways...

- `kOfxParamDoubleTypeX`
size in the X dimension, in canonical coords (1D double only),
- `kOfxParamDoubleTypeXAbsolute`
positing in the X axis, in canonical coords (1D double only)
- `kOfxParamDoubleTypeY`
size in the Y dimension, in canonical coords (1D double only),
- `kOfxParamDoubleTypeYAbsolute`
positing in the Y axis, in canonical coords (1D double only)
- `kOfxParamDoubleTypeXY`
2D size, in canonical coords (2D double only),
- `kOfxParamDoubleTypeXYAbsolute`
2D position, in canonical coords. (2D double only).

Spatial Normalised Double Parameters

Ideally, normalised parameters should be deprecated and no longer used if spatial parameters are available.

There are several values of the *kOfxParamPropDoubleType* that say it should be interpreted as a size or position. These are expressed and proportional to the current project's size. This will allow the parameter to scale cleanly with project size changes and to be represented to the user in an appropriate range.

For example, the sensible X range of a visual effect plugin is the project's width, say 768 pixels for a PAL D1 definition video project. The user sees the parameter as 0..768, the effect sees it as 0..1. So if the plug-in wanted to set the default value of an effect to be the centre of the image, it would flag a 2D parameter as normalised and set the defaults to be 0.5. The user would see this in the centre of the image, no matter the resolution of the project in question. The plugin would retrieve the parameter as 0..1 and scale it up to the project size to size to use.

Parameters can choose to be normalised in several ways...

- *kOfxParamDoubleTypeNormalisedX*
normalised size wrt to the project's X dimension (1D only),
- *kOfxParamDoubleTypeNormalisedXAbsolute*
normalised absolute position on the X axis (1D only)
- *kOfxParamDoubleTypeNormalisedY*
normalised size wrt to the project's Y dimension(1D only),
- *kOfxParamDoubleTypeNormalisedYAbsolute*
normalised absolute position on the Y axis (1D only)
- *kOfxParamDoubleTypeNormalisedXY*
normalised to the project's X and Y size (2D only),
- *kOfxParamDoubleTypeNormalisedXYAbsolute*
normalised to the projects X and Y size, and is an absolute position on the image plane.

See the section on coordinate systems on how to scale between normalised, canonical and pixel coordinates.

1.12.32 Double Parameters Defaults, Increments, Mins and Maxs

In all cases double parameters' defaults, minimums and maximums are specified in the same space as the parameter, as is the increment in all cases but normalised parameters.

Normalised parameters specify their increments in canonical coordinates, rather than in normalised coordinates. So an increment of '1' means 1 pixel, not '1 project width', otherwise sliders would be a bit wild.

1.12.33 Parametric Parameters

Introduction

Parametric params are new for 1.2 and are optionally supported by host applications. They are specified via the *kOfxParamTypeParametric* identifier passed into *OfxParameterSuiteV1::paramDefine()*

These parameters are somewhat more complex than normal parameters and require their own set of functions to manage and manipulate them. The new *OfxParametricParameterSuiteV1* is there to do that.

All the defines and suite definitions for parametric parameters are defined in the file *ofxParametricParam.h*

Parametric parameters are in effect *functions* a plug-in can ask a host to arbitrarily evaluate for some value *x*. A classic use case would be for constructing look-up tables, a plug-in would ask the host to evaluate one at multiple values from 0 to 1 and use that to fill an array.

A host would probably represent this to a user as a cubic curve in a standard curve editor interface, or possibly through scripting. The user would then use this to define the ‘shape’ of the parameter.

The evaluation of such params is not the same as animation, they are returning values based on some arbitrary argument orthogonal to time, so to evaluate such a param, you need to pass a parametric position and time.

Often, you would want such a parametric parameter to be multi-dimensional, for example, a colour look-up table might want three values, one for red, green and blue. Rather than declare three separate parametric parameters, so a parametric parameter can be multi-dimensional.

Due to the nature of the underlying data, you *cannot* call certain functions in the ordinary parameter suite when manipulating a parametric parameter. All functions in the standard parameter suite are valid when called on a parametric parameter, with the exception of the following....

- `OfxParameterSuiteV1::paramGetValue()`
- `OfxParameterSuiteV1::paramGetValueAtTime()`
- `OfxParameterSuiteV1::paramGetDerivative()`
- `OfxParameterSuiteV1::paramGetIntegral()`
- `OfxParameterSuiteV1::paramSetValue()`
- `OfxParameterSuiteV1::paramSetValueAtTime()`

Defining Parametric Parameters

Parametric parameters are defined using the standard parameter suite function `OfxParameterSuiteV1::paramDefine()`. The descriptor returned by this call have several non standard parameter properties available. These are

- `kOfxParamPropParametricDimension`
the dimension of the parametric parameter,
- `kOfxParamPropParametricUIColor`
the colour of the curves of a parametric parameter in any user interface
- `kOfxParamPropParametricInteractBackground`
a pointer to an interact entry point, which will be used to draw a background under any user interface,
- `kOfxParamPropParametricRange`
the min and max value that the parameter will be evaluated over.

Animating Parametric Parameters

Animation is an optional host feature for parametric parameters. Hosts flag whether they support this feature by setting the host descriptor property `kOfxParamHostPropSupportsParametricAnimation`.

Getting and Setting Values on a Parametric Parameters

Seeing as we need to pass in the parametric position and dimension to evaluate, parametric parameters need a new evaluation mechanism. They do this with the `OfxParametricParameterSuiteV1::parametricParamGetValue()` function. This function returns the value of the parameter at the given time, for the given dimension, and the given parametric position.

Parametric parameters are effectively interfaces to some sort of host based curve library. To get/set/delete points in the curve that represents a parameter, the new suite has several functions available to manipulate control points of the underlying curve.

To set the default value of a parametric parameter to anything but the identity, you use the control point setting functions in the new suite to set up a curve on the *descriptor* returned by `OfxParameterSuiteV1::paramDefine()`. Any instances later created, will have that curve as a default.

Example

This simple example defines a colour lookup table, defines a default, and show how to evaluate the curve

```
// describe our parameter in
static OfxStatus
describeInContext( OfxImageEffectHandle effect, OfxPropertySetHandle inArgs)
{
    ....
    // define it
    OfxPropertySetHandle props;
    gParamHost->paramDefine(paramSet, kOfxParamTypeParametric, "lookupTable", &
->props);

    // set standard names and labels
    gPropHost->propSetString(props, kOfxParamPropHint, 0, "Colour lookup table");
    gPropHost->propSetString(props, kOfxParamPropScriptName, 0, "lookupTable");
    gPropHost->propSetString(props, kOfxPropLabel, 0, "Lookup Table");

    // define it as three dimensional
    gPropHost->propSetInt(props, kOfxParamPropParametricDimension, 0, 3);

    // label our dimensions are r/g/b
    gPropHost->propSetString(props, kOfxParamPropDimensionLabel, 0, "red");
    gPropHost->propSetString(props, kOfxParamPropDimensionLabel, 1, "green");
    gPropHost->propSetString(props, kOfxParamPropDimensionLabel, 2, "blue");

    // set the UI colour for each dimension
    for(int component = 0; component < 3; ++component) {
        gPropHost->propSetDouble(props, kOfxParamPropParametricUIColor,
->component * 3 + 0, component % 3 == 0 ? 1 : 0);
        gPropHost->propSetDouble(props, kOfxParamPropParametricUIColor,
->component * 3 + 1, component % 3 == 1 ? 1 : 0);
        gPropHost->propSetDouble(props, kOfxParamPropParametricUIColor,
->component * 3 + 2, component % 3 == 2 ? 1 : 0);
    }

    // set the min/max parametric range to 0..1
    gPropHost->propSetDouble(props, kOfxParamPropParametricRange, 0, 0.0);
```

(continues on next page)

(continued from previous page)

```

gPropHost->propSetDouble(props, kOfxParamPropParametricRange, 1, 1.0);

// set a default curve, this example sets an invert
OfxParamHandle descriptor;
gParamHost->paramGetHandle(paramSet, "lookupTable", &descriptor, NULL);
for(int component = 0; component < 3; ++component) {
    // add a control point at 0, value is 1
    gParametricParamHost->parametricParamAddControlPoint(descriptor,
        component, // curve
        0.0, // time,
        0.0, // parametric
        1.0, // value to
        false); // don't
    // add a key
    // add a control point at 1, value is 0
    gParametricParamHost->parametricParamAddControlPoint(descriptor, component,
        0.0, 1.0, 0.0, false);
}
...
}

void render8Bits(double currentFrame, otherStuff...)
{
    ...

    // make three luts from our curves
    unsigned char lut[3][256];

    OfxParamHandle param;
    gParamHost->paramGetHandle(paramSet, "lookupTable", &param, NULL);
    for(int component = 0; component < 3; ++component) {
        for(int position = 0; position < 256; ++position) {
            // position to evaluate the param at
            float parametricPos = float(position)/255.0f;

            // evaluate the parametric param
            float value;
            gParametricParamHost->parametricParamGetValue(param, component,
                currentFrame, parametricPos, &value);
            value = value * 255;
            value = clamp(value, 0, 255);

            // set that in the lut
            lut[dimension][position] = (unsigned char)value;
        }
    }
    ...
}

```

(continues on next page)

(continued from previous page)

}

1.12.34 Setting Parameters

Plugins are free to set parameters in limited set of circumstances, typically relating to user interaction. You can only set parameters in the following actions passed to the plug-in's main entry function...

- *kOfxActionCreateInstance*
- *kOfxActionBeginInstanceChanged*
- *kOfxActionInstanceChanged*
- *kOfxActionEndInstanceChanged*
- *kOfxActionSyncPrivateData*

Plugins can also set parameter values during the following actions passed to any of its interacts main entry function:

- *kOfxInteractActionPenDown*
- *kOfxInteractActionPenMotion*
- *kOfxInteractActionPenUp*
- *kOfxInteractActionKeyDown*
- *kOfxInteractActionKeyRepeat*
- *kOfxInteractActionKeyUp*
- *kOfxInteractActionLoseFocus*

1.13 Rendering

The *kOfxImageEffectActionRender* action is passed to plugins when the host requires them to render an output frame.

All calls to the *kOfxImageEffectActionRender* are bracketed by a pair of *kOfxImageEffectActionBeginSequenceRender* and *kOfxImageEffectActionEndSequenceRender* actions. This is to allow plugins to prepare themselves for rendering long sequences by setting up any tables etc.. it may need.

The *kOfxImageEffectActionBeginSequenceRender* will indicate the frame range that is to be rendered, and whether this is purely a single frame render due to interactive feedback from a user in a GUI.

1.13.1 Identity Effects

If an effect does nothing to its input clips (for example a blur with blur size set to '0') it can indicate that it is an identity function via the *kOfxImageEffectActionIsIdentity* action. The plugin indicates which input the host should use for the region in question. This allows a host to short circuit the processing of an effect.

1.13.2 Rendering and The Get Region Actions

Many hosts attempt to minimise the areas that they render by using regions of interest and regions of definition, while some of the simpler hosts do not attempt to do so. In general the order of actions, per frame rendered, is something along the lines of...

- ask the effect for it's region of definition,
- clip the render window against that
- ask the effect for the regions of interest of each of it's inputs against the clipped render window,
- clip those regions of interest against the region of definition of each of those inputs,
- render and cache each of those inputs,
- render the effect against it's clipped render window.

A host can ask an effect to render an arbitrary window of pixels, generally these should be clipped to an effect's region of definition, however, depending on the host, they may not be. The actual region to render is indicated by the *kOfxImageEffectPropRenderWindow* render action argument. If an effect is asked to render outside of its region of definition, it should fill those pixels in with black transparent pixels.

Note that *OfxImageEffectSuiteV1::clipGetImage()* function takes an optional *region* parameter. This is a region, in Canonical coordinates, that the effect would like on that input clip. If not used in a render action, then the image returned should be based on the previous get region of interest action. If used, then the image returned will be based on this (usually be clipped to the input's region of definition). Generally a plugin should not use the *region* parameter in the render action, but should leave it to the 'default' region.

1.13.3 Multi-threaded Rendering

Multiple render actions may be passed to an effect at the same time. A plug-in states it's level of render thread safety by setting the *kOfxImageEffectPluginRenderThreadSafety* string property. This can be set to one of three states...

kOfxImageEffectRenderUnsafe

String used to label render threads as un thread safe, see, *kOfxImageEffectPluginRenderThreadSafety*.

Indicating that only a single 'render' action can be made at any time among all instances

kOfxImageEffectRenderInstanceSafe

String used to label render threads as instance thread safe, *kOfxImageEffectPluginRenderThreadSafety*.

Indicating that any instance can have a single 'render' action at any one time

kOfxImageEffectRenderFullySafe

String used to label render threads as fully thread safe, *kOfxImageEffectPluginRenderThreadSafety*.

Indicating that any instance of a plugin can have multiple renders running simultaneously

Rendering in a Symmetric Multi Processing Environment

When rendering on computers that have more than one CPU (or this new-fangled hyperthreading), hosts and effects will want to take advantage of all that extra CPU goodness to speed up rendering. This means multi-threading of the render function in some way.

If the plugin has set *kOfxImageEffectPluginRenderThreadSafety* to *kOfxImageEffectRenderFullySafe*, the host may choose to render a single frame across multiple CPUs by having each CPU render a different window. However, the plugin may wish to remain in charge of multithreading a single frame. The plugin set property *kOfxImageEffectPluginPropHostFrameThreading* informs the host as to whether the host should perform SMP on the effect. It can be set to either...

- 1, in which case the host will attempt to multithread an effect instance by calling its render function called simultaneously, each call will be with a different renderWindow, but be at the same frame
- 0, in which case the host only ever calls the render function once per frame. If the effect wants to multithread it must use the *OfxMultiThreadSuite* API.

A host may have a render farm of computers. Depending exactly how the host works with its render farm, it may have multiple copies on an instance spread over the farm rendering separate frame ranges, 1-100 on station A, 101 to 200 on station B and so on...

Rendering Sequential Effects

Some plugins need the output of the previous frame to render the next, typically they cache some information about the last render and use that somehow on the next frame. Some temporally averaging degrading algorithms work that way. Such effects cannot render correctly unless they are strictly rendered in order, from first to last frame, on a single instance.

Other plugins are able to render correctly when called in an arbitrary frame order, but render much more efficiently if rendered in order. For example a particle system which maintains the state of the particle system in an instance would simply increment the simulation by a frame if rendering in-order, but would need to restart the particle system from scratch if the frame jumped backwards.

Most plug-ins do not have any sequential dependence. For example, a simple gain operation has no dependence on the previous frame.

Similarly, host applications, due to their architectures, may or may not be able to guarantee that a plugin can be rendered strictly in-order. Node based applications typically have much more difficulty in guaranteeing such behaviour.

To indicate whether a plugin needs to be rendered in a strictly sequential order, and to indicate whether a host supports such behaviour we have a property, *kOfxImageEffectInstancePropSequentialRender*. For plug-ins this can be one of three values...

- 0, in which case the host can render an instance over arbitrary frame ranges on an arbitrary number of computers without any problem (default),
- 1, in which case the host must render an instance on a single computer over its entire frame range, from first to last.
- 2, in which case the effect is more efficiently rendered in frame order, but can compute the correct result regardless of render order.

For hosts, this property takes three values...

- 0, which indicates that the host can never guarantee sequential rendering,
- 1, which indicates that the host can guarantee sequential rendering for plugins that request it,
- 2, which indicates that the host can sometimes perform sequential rendering.

When rendering, a host will set the `in args` property on `kOfxImageEffectPropSequentialRenderStatus` to indicate whether the host is currently supporting sequential renders. This will be passed to the following actions,

- the begin sequence render action
- the sequence render action
- the end sequence render action

Hosts may still render sequential effects with random frame access in interactive sessions, for example when the user scrubs the current frame on the timeline and the host asks an effect to render a preview frame. In such cases, the plugin can detect that the instance is being interactively manipulated via the `kOfxImageEffectPropInteractiveRenderStatus` property and hack an approximation together for UI purposes. If eventually rendering the sequence, the host *must* ignore all frames rendered out of order and not cache them for use in the final result.

A host may set the `in args` property `kOfxImageEffectPropRenderQualityDraft` in `:c:macro:kOfxImageEffectActionRender` to ask for a render in Draft/Preview mode. This is useful for applications that must support fast scrubbing. These allow a plug-in to take short-cuts for improved performance when the situation allows and it makes sense, for example to generate thumbnails with effects applied. For example switch to a cheaper interpolation type or rendering mode. A plugin should expect frames rendered in this manner that will not be stuck in host cache unless the cache is only used in the same draft situations.`

1.13.4 OFX : Fields and Field Rendering

Fields are evil, but until the world decides to adopt sensible video standard and casts the current ones into the same pit as 2 inch video tape, we are stuck with them.

Before we start, some nomenclature. The Y-Axis is considered to be up, so in a fielded image,

- even scan lines 0,2,4,6,... are collectively referred to as the lower field,
- odd scan lines 1,3,5,7... are collective referred to as the upper field.

We don't call them odd and even, so as to avoid confusion with video standard, which have scanline 0 at the top, and so have the opposite sense of our 'odd' and 'even'.

Clips and images from those clips are flagged as to whether they are fielded or not, and if so what is the spatial/temporal ordering of the fields in that image. The `kOfxImageClipPropFieldOrder` clip and image instance property can be...

kOfxImageFieldNone

String used to label imagery as having no fields

kOfxImageFieldLower

String used to label the lower field (scan lines 0,2,4...) of fielded imagery

kOfxImageFieldUpper

String used to label the upper field (scan lines 1,3,5...) of fielded imagery

Images extracted from a clip flag what their fieldedness is with the property `kOfxImagePropField`, this can be...

kOfxImageFieldNone

String used to label imagery as having no fields

kOfxImageFieldBoth

String used to label both fields of fielded imagery, indicating interlaced footage

kOfxImageFieldLower

String used to label the lower field (scan lines 0,2,4...) of fielded imagery

kOfxImageFieldUpper

String used to label the upper field (scan lines 1,3,5...) of fielded imagery

The plugin specifies how it deals with fielded imagery by setting this property:

kOfxImageEffectPluginPropFieldRenderTwiceAlways

Controls how a plugin renders fielded footage.

- Type - integer X 1
- Property Set - a plugin descriptor (read/write)
- Default - 1
- Valid Values - This must be one of
 - 0 - the plugin is to have its render function called twice, only if there is animation in any of its parameters
 - 1 - the plugin is to have its render function called twice always

The reason for this is an optimisation. Imagine a text generator with no animation being asked to render into a fielded output clip, it can treat an interlaced fielded image as an unfielded frame. So the host can get the effect to render both fields in one hit and save on the overhead required to do the rendering in two passes.

If called twice per frame, the time passed to the render action will be frame and frame+0.5. So 0.0 0.5 1.0 1.5 etc...

When rendering unfielded footage, the host will only ever call the effect's render action once per frame, with the time being at the integers, 0.0, 1.0, 2.0 and so on.

The render action's argument property *kOfxImageEffectPropFieldToRender* tells the effect which field it should render, this can be one of...

- *kOfxImageFieldNone* - there are no fields to deal with, the image is full frame
- *kOfxImageFieldBoth* - the imagery is fielded and both scan lines should be rendered
- *kOfxImageFieldLower* - the lower field is being rendered (lines 0,2,4...)
- *kOfxImageFieldUpper* - the upper field is being rendered (lines 1,3,5...)

Note: *kOfxImageEffectPropFieldToRender* will be set to *kOfxImageFieldBoth* if *kOfxImageEffectPluginPropFieldRenderTwiceAlways* is set to 0 on the plugin

A plugin can specify how it wishes fielded footage to be fetched from a clip via the clip descriptor property *kOfxImageClipPropFieldExtraction*. This can be one of...

- *kOfxImageFieldBoth*

Fetch a full frame interlaced image

- *kOfxImageFieldSingle*

Fetch a single field, making a half height image

- `kOfxImageFieldDoubled`

Fetch a single field, but doubling each line and so making a full height image (default)

If fetching a single field, the actual field fetched from the source frame is...

- the first temporal field if the time passed to `clipGetImage` has a fractional part of $0.0 \leq f < 0.5$
- the second temporal field otherwise,

To illustrate this last behaviour, the two examples below show an output with twice the frame rate of the input and how `clipGetImage` maps to the input. The `.0` and `.5` mean first and second temporal fields.

Behaviour **with** unfielded footage

```
output 0      1      2      3
source 0      0      1      1
```

Behaviour **with** fielded footage

```
output 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5
source 0.0 0.0 0.5 0.5 1.0 1.0 1.5 1.5
```

NOTE

- while some rarely used video standards can have odd number of scan-lines, under OFX, both fields always consist of the same number of lines. Pad with black where needed.
- host developers, for single field extracted images, you don't need to do any buffer copies, you just need to set the `row bytes` property of the returned image to twice the normal value, and maybe tweak the start address by a scanline.

1.13.5 Rendering In An Interactive Environment

Any host with an interface will most likely have an interactive thread and a rendering thread. This allows an effect to be manipulated while having renders batched off to a background thread. This will mean that some degree of locking will go on to prevent simultaneous read/writes occurring, see this section for more on thread safety.

A host may need to abort a backgrounded render, typically in response to a user changing a parameter value. An effect should occasionally poll the `OfxImageEffectSuiteV1::abort()` function to see if it should give up on rendering.

1.13.6 Rendering on GPU

page `ofxOpenGLRender`

Introduction

The `OfxOpenGLRenderSuite` allows image effects to use OpenGL commands (hopefully backed by a GPU) to accelerate rendering of their outputs. The basic scheme is simple...

- An effect indicates it wants to use OpenGL acceleration by setting the `kOfxImageEffectPropOpenGLRenderSupported` flag on its descriptor
- A host indicates it supports OpenGL acceleration by setting `kOfxImageEffectPropOpenGLRenderSupported` on its descriptor
- In an effect's `kOfxImageEffectActionGetClipPreferences` action, an effect indicates what clips it will be loading images from onto the GPU's memory during an effect's `kOfxImageEffectActionRender` action.

OpenGL House Keeping

If a host supports OpenGL rendering then it flags this with the string property `kOfxImageEffectPropOpenGLRenderSupported` on its descriptor property set. Effects that cannot run without OpenGL support should examine this in `kOfxActionDescribe` action and return a `kOfxStatErrMissingHostFeature` status flag if it is not set to "true".

Effects flag to a host that they support OpenGL rendering by setting the string property `kOfxImageEffectPropOpenGLRenderSupported` on their effect descriptor during the `kOfxActionDescribe` action. Effects can work in three ways...

- purely on CPUs without any OpenGL support at all, in which case they should set `kOfxImageEffectPropOpenGLRenderSupported` to be "false" (the default),
- on CPUs but with optional OpenGL support, in which case they should set `kOfxImageEffectPropOpenGLRenderSupported` to be "true",
- only with OpenGL support, in which case they should set `kOfxImageEffectPropOpenGLRenderSupported` to be "needed".

Hosts can examine this flag and respond to it appropriately.

Effects can use OpenGL accelerated rendering during the following action...

- `kOfxImageEffectActionRender`

If an effect has indicated that it optionally supports OpenGL acceleration, it should check the property `kOfxImageEffectPropOpenGLEnabled` passed as an argument to the following actions,

- `kOfxImageEffectActionRender`
- `kOfxImageEffectActionBeginSequenceRender`
- `kOfxImageEffectActionEndSequenceRender`

If this property is set to 0, then it should not attempt to use any calls to the OpenGL suite or OpenGL calls whilst rendering.

Getting Images as Textures

An effect could fetch an image into memory from a host via the standard Image Effect suite “clipGetImage” call, then create an OpenGL texture from that. However as several buffer copies and various other bits of house keeping may need to happen to do this, it is more efficient for a host to create the texture directly.

The `OfxOpenGLRenderSuiteV1::clipLoadTexture` function does this. The arguments and semantics are similar to the `OfxImageEffectSuiteV2::clipGetImage` function, with a few minor changes.

The effect is passed back a property handle describing the texture. Once the texture is finished with, this should be disposed of via the `OfxOpenGLRenderSuiteV1::clipFreeTexture` function, which will also delete the associated OpenGL texture (for source clips).

The returned handle has a set of properties on it, analogous to the properties returned on the image handle by `OfxImageEffectSuiteV2::clipGetImage`. These are:

- *kOfxImageEffectPropOpenGLTextureIndex*
- *kOfxImageEffectPropOpenGLTextureTarget*
- *kOfxImageEffectPropPixelDepth*
- *kOfxImageEffectPropComponents*
- *kOfxImageEffectPropPreMultiplication*
- *kOfxImageEffectPropRenderScale*
- *kOfxImagePropPixelAspectRatio*
- *kOfxImagePropBounds*
- *kOfxImagePropRegionOfDefinition*
- *kOfxImagePropRowBytes*
- *kOfxImagePropField*
- *kOfxImagePropUniqueIdentifier*

The main difference between this and an image handle is that the *kOfxImagePropData* property is replaced by the *kOfxImageEffectPropOpenGLTextureIndex* property. This integer property should be cast to a `GLuint` and is the index to use for the OpenGL texture. Next to texture handle the texture target enumerator is given in *kOfxImageEffectPropOpenGLTextureTarget*

Note, because the image is being directly loaded into a texture by the host it need not obey the Clip Preferences action to remap the image to the pixel depth the effect requested.

Render Output Directly with OpenGL

Effects can use the graphics context as they see fit. They may be doing several render passes with fetch back from the card to main memory via ‘render to texture’ mechanisms interleaved with passes performed on the CPU. The effect must leave output on the graphics card in the provided output image texture buffer.

The host will create a default OpenGL viewport that is the size of the render window passed to the render action. The following code snippet shows how the viewport should be rooted at the bottom left of the output texture.

```
// set up the OpenGL context for the render to texture
...

// figure the size of the render window
int dx = renderWindow.x2 - renderWindow.x1;
```

(continues on next page)

(continued from previous page)

```
int dy = renderWindow.y2 - renderWindow.y2;

// setup the output viewport
glViewport(0, 0, dx, dy);
```

Prior to calling the render action the host may also choose to bind the output texture as the current color buffer (render target), or they may defer doing this until `clipLoadTexture` is called for the output clip.

After this, it is completely up to the effect to choose what OpenGL operations to render with, including projections and so on.

OpenGL Current Context

The host is only required to make the OpenGL context current (e.g., using `wglMakeCurrent`, for Windows) during the following actions:

- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*
- `kOfxActionOpenGLContextAttached`
- `kOfxActionOpenGLContextDetached`

For the first 3 actions, Render through `EndSequenceRender`, the host is only required to set the OpenGL context if *kOfxImageEffectPropOpenGLEnabled* is set. In other words, a plug-in should not expect the OpenGL context to be current for other OFX calls, such as *kOfxImageEffectActionDescribeInContext*.

group **CudaRender**

Version

CUDA rendering was added in version 1.5.

Defines

kOfxImageEffectPropCudaRenderSupported

Indicates whether a host or plug-in can support CUDA render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support CUDA render
 - “true” - the host or plug-in can support CUDA render

kOfxImageEffectPropCudaEnabled

Indicates that a plug-in SHOULD use CUDA render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropCudaRenderSupported="true"` then the host MAY set this property to indicate that it is passing images as CUDA memory pointers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the `kOfxImagePropData` of each image of each clip is a CPU memory pointer.
 - 1 indicates that the `kOfxImagePropData` of each image of each clip is a CUDA memory pointer.

kOfxImageEffectPropCudaStreamSupported

Indicates whether a host or plug-in can support CUDA streams.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - "false" for a plug-in
- Valid Values - This must be one of
 - "false" - in which case the host or plug-in does not support CUDA streams
 - "true" - which means a host or plug-in can support CUDA streams

kOfxImageEffectPropCudaStream

The CUDA stream to be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property will only be set if the host and plug-in both support CUDA streams.

If set:

- this property contains a pointer to the stream of CUDA render (`cudaStream_t`). In order to use it, `reinterpret_cast<cudaStream_t>(pointer)` is needed.

- the plug-in SHOULD ensure that its render action enqueues any asynchronous CUDA operations onto the supplied queue.
- the plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action, and SHOULD NOT call `cudaDeviceSynchronize()` at any time.

If not set:

- the plug-in SHOULD ensure that any asynchronous operations it enqueues have completed before returning from the render action.

group **MetalRender**

Version

Metal rendering was added in version 1.5.

Defines

kOfxImageEffectPropMetalRenderSupported

Indicates whether a host or plug-in can support Metal render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support Metal render
 - “true” - the host or plug-in can support Metal render

kOfxImageEffectPropMetalEnabled

Indicates that a plug-in SHOULD use Metal render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropMetalRenderSupported=“true”` then the host MAY set this property to indicate that it is passing images as Metal buffers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the `kOfxImagePropData` of each image of each clip is a CPU memory pointer.
 - 1 indicates that the `kOfxImagePropData` of each image of each clip is a Metal `id<MTLBuffer>`.

kOfxImageEffectPropMetalCommandQueue

The command queue of Metal render.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property contains a pointer to the command queue to be used for Metal rendering (id<MTLCommandQueue>). In order to use it, reinterpret_cast<id<MTLCommandQueue>>(pointer) is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous Metal operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

group **OpenCLRender****Version**

OpenCL rendering was added in version 1.5.

Defines**kOfxImageEffectPropOpenCLRenderSupported**

Indicates whether a host or plug-in can support OpenCL Buffers render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support OpenCL Buffers render
 - “true” - the host or plug-in can support OpenCL Buffers render

kOfxImageEffectPropOpenCLSupported

Indicates whether a host or plug-in can support OpenCL Images render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of

- "false" - in which case the host or plug-in does not support OpenCL Images render
- "true" - which means a host or plug-in can support OpenCL Images render

kOfxImageEffectPropOpenCLEnabled

Indicates that a plug-in SHOULD use OpenCL render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropOpenCLRenderSupported="true"` or have both set `kOfxImageEffectPropOpenCLSupported="true"` then the host MAY set this property to indicate that it is passing images as OpenCL Buffers or Images.

When rendering using OpenCL Buffers, the `cl_mem` of the buffers are retrieved using *kOfxImagePropData*. When rendering using OpenCL Images, the `cl_mem` of the images are retrieved using *kOfxImageEffectPropOpenCLImage*. If both *kOfxImageEffectPropOpenCLSupported* (Buffers) and *kOfxImageEffectPropOpenCLRenderSupported* (Images) are enabled by the plug-in, it should use *kOfxImageEffectPropOpenCLImage* to determine which is being used by the host.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that a plug-in SHOULD use OpenCL render in the render action
 - 1 indicates that a plug-in SHOULD NOT use OpenCL render in the render action

kOfxImageEffectPropOpenCLCommandQueue

Indicates the OpenCL command queue that should be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property contains a pointer to the command queue to be used for OpenCL rendering (`cl_command_queue`). In order to use it, `reinterpret_cast<cl_command_queue>(pointer)` is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous OpenCL operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

kOfxImageEffectPropOpenCLImage

Indicates the image handle of an image supplied as an OpenCL Image by the host.

- Type - pointer X 1
- Property Set - image handle returned by clipGetImage

This value should be cast to a `cl_mem` and used as the image handle when performing OpenCL Images operations. The property should be used (not `kOfxImagePropData`) when rendering with OpenCL Images (`kOfxImageEffectPropOpenCLSupported`), and should be used to determine whether Images or Buffers should be used if a plug-in supports both `kOfxImageEffectPropOpenCLSupported` and `kOfxImageEffectPropOpenCLRenderSupported`. Note: the `kOfxImagePropRowBytes` property is not required to be set by the host, since OpenCL Images do not have the concept of row bytes.

kOfxOpenCLProgramSuite

Typedefs

typedef struct *OfxOpenCLProgramSuiteV1* **OfxOpenCLProgramSuiteV1**

OFX suite that allows a plug-in to get OpenCL programs compiled.

This is an optional suite the host can provide for building OpenCL programs for the plug-in, as an alternative to calling `clCreateProgramWithSource / clBuildProgram`. There are two advantages to doing this: The host can add flags (such as `-cl-denorms-are-zero`) to the build call, and may also cache program binaries for performance (however, if the source of the program or the OpenCL environment changes, the host must recompile so some mechanism such as hashing must be used).

struct **OfxOpenCLProgramSuiteV1**

#include <ofxGPURender.h> OFX suite that allows a plug-in to get OpenCL programs compiled.

This is an optional suite the host can provide for building OpenCL programs for the plug-in, as an alternative to calling `clCreateProgramWithSource / clBuildProgram`. There are two advantages to doing this: The host can add flags (such as `-cl-denorms-are-zero`) to the build call, and may also cache program binaries for performance (however, if the source of the program or the OpenCL environment changes, the host must recompile so some mechanism such as hashing must be used).

Public Members

OfxStatus (***compileProgram**)(const char *pszProgramSource, int fOptional, void *pResult)

Compiles the OpenCL program.

1.14 Interacts

When a host presents a graphical user interface to an image effect, it may optionally give it the chance to draw its own custom GUI tools and to be able to interact with pen and keyboard input. In OFX this is done via the `OfxInteract` suite, which is found in the file `ofxInteract.h`.

OFX interacts by default use OpenGL to perform all drawing in interacts, due to its portability, robustness and wide implementation. As of 2022, some systems are moving away from OpenGL support in favor of more modern graphics drawing APIs. So as of OFX 1.5, effects may use the `OfxDrawSuiteV1` instead of OpenGL if the host supports it.

Each object that can have their own interact a pointer property in it which should point to a separate *main entry point*. This entry point is *not* the same as the one in the `OfxPlugin` struct, as it needs to respond to a different set of actions to the effect.

There are two things in an image effect can have their own interact, these are...

- as an overlay on the image being currently viewed in any image viewer, set via the effect descriptor's *kOfxImageEffectPluginPropOverlayInteractV1* property
- as a replacement for any parameter's standard GUI object, set this via the parameter descriptor's *kOfiParamPropInteractV1* property.

Hosts might not be able to support interacts, to indicate this, two properties exist on the host descriptor which an effect should examine at description time so as to determine its own behaviour. These are...

- *kOfxImageEffectPropSupportsOverlays*
- *kOfiParamHostPropSupportsCustomInteract*

Interacts are separate objects to the effect they are associated with, they have their own descriptor and instance handles passed into their separate *main entry point*.

An interact instance cannot exist without a plugin instance, an interact's instance, once created, is bound to a single instance of a plugin until the interact instance is destroyed.

All interacts of the same type share openGL display lists, even if they are in different openGL contexts.

All interacts of the same type will have the same pixel types (this is a side effect of the last point), this will always be double buffered with at least RGB components. Alpha and the exact bit depth is left to the implementation.

So for example, all image effect overlays share the same display lists and have the same pixel depth, and all custom parameter GUIs share the same display list and have the same pixel depth, but overlays and custom parameter GUIs do not necessarily share the same display list/pixel depths.

An interact instance may be used in more than one view. Consider an image effect overlay interact in a host that supports multiple viewers to an effect instance. The same interact instance will be used in all views, the relevant properties controlling the view being changed before any action is passed to the interact. In this example, the draw action would be called once for each view open on the instance, with the projection, viewport and pixel scale being set appropriately for the view before each action.

1.14.1 Overlay Interacts

Hosts will generally display images (both input and output) in user their interfaces. A plugin can put an interact in this display by setting the effect descriptor *kOfxImageEffectPluginPropOverlayInteractV1* property to point to a main entry.

The viewport for such interacts will depend completely on the host.

The GL_PROJECTION matrix will be set up so that it maps openGL coordinates to canonical image coordinates.

The GL_MODELVIEW matrix will be the identity matrix.

An overlay's interact draw action should assume that it is sharing the openGL context and viewport with other objects that belong to the host. It should not blank the background and it should never swap buffers, that is for the host to do.

1.14.2 Parameter Interacts

All parameters, except for custom parameters, have some default interface that the host creates for them. Be it a numeric slider, colour swatch etc... Effects can override the default interface (or set an interface for a custom parameter) by setting the *kOfxParamPropInteractV1*. This will completely replace the parameters default user interface in the 'paged' and *hierarchical* interfaces, but it will not replace the parameter's interface in any animation sheet.

Properties affecting custom interacts for parameters are...

- *kOfxParamPropInteractSizeAspect*
- *kOfxParamPropInteractMinimumSize*
- *kOfxParamPropInteractPreferredSize*

The viewport for such interacts will be dependent upon the various properties above, and possibly a per host override in any XML resource file.

The *GL_PROJECTION* matrix will be an orthographic 2D view with -0.5,-0.5 at the bottom left and viewport width-0.5, viewport height-0.5 at the top right.

The *GL_MODELVIEW* matrix will be the identity matrix.

The bit depth will be double buffered 24 bit RGB.

A parameter's interact draw function will have full responsibility for drawing the interact, including clearing the background and swapping buffers.

1.14.3 Interact Actions

The following actions are passed to any interact entry point in an image effect plug-in.

- The Generic Describe Action called to describe the specific interact ,
- The Create Instance Action called just after an instance of the interact is created,
- The Generic Destroy Instance Action called just before of the interact is destroyed,
- The Draw Action called to have the interact draw itself,
- *kOfxInteractActionPenMotion* called whenever the interact has the input focus and the pen has moved, regardless of whether the pen is up or down,
- *kOfxInteractActionPenDown* called whenever the interact has the input focus and the pen has changed state to 'down',
- *kOfxInteractActionPenUp* called whenever the interact has the input focus and the pen has changed state to 'up',
- *kOfxInteractActionKeyDown* called whenever the interact has the input focus and a key has gone down,
- *kOfxInteractActionKeyUp* called whenever the interact has the input focus and a key has gone up,
- *kOfxInteractActionKeyRepeat* called whenever the interact has the input focus and a key has gone down and a repeat key sequence has been sent,
- *kOfxInteractActionGainFocus* called whenever the interact gains input focus,
- *kOfxInteractActionLoseFocus* called whenever the interact loses input focus,

An interact cannot be described until an effect has been described.

An interact instance must always be associated with an effect instance. So it gets created after an effect and destroyed before one.

An interact instance should be issued a gain focus action before any key or pen actions are issued, and a lose focus action when it goes.

1.15 Image Effect Clip Preferences

The *kOfxImageEffectActionGetClipPreferences* action is passed to an effect to allow a plugin to specify how it wishes to deal with its input clips and to set properties in its output clip. This is especially important when there are multiple inputs which may have differing properties such as pixel depth and number of channels.

More specifically, there are six properties that can be set during the clip preferences action, some on the input clip, some on the output clip, some on both. These are:

- the depth of a clip's pixels, input or output clip
- the components of a clip's pixels, input or output clip
- the pixel aspect ratio of a clip, input or output clip
- the frame rate of the output clip
- the fielding of the output clip
- the premultiplication state of the output clip
- whether the output clip varies from frame to frame, even if no parameters or input images change over time
- whether the output clip can be sampled at sub-frame times and produce different images

The behaviour specified by OFX means that a host may need to cast images from their native data format into one suitable for the plugin. It is better that the host do any of this pixel shuffling because:

- the behaviour is orthogonal for all plugins on that host
- the code is not replicated in all plugins
- the host can optimise the pixel shuffling in one pass with any other data grooming it may need to do

A plugin gets to assert its clip preferences in several situations. Firstly whenever a clip is attached to a plugin, secondly whenever one of the parameters in the plugin property *kOfxImageEffectPropClipPreferencesSlaveParam* has its value changed. The clip preferences action is never called until all non-optional clips have been attached to the plugin.

Note:

- these properties cannot animate over the duration of an effect
 - that the ability to set input and output clip preferences is restricted by the context of an effect
 - optional input clips do not have any context specific restrictions on plugin set preferences
-

1.15.1 Frame Varying Effects

Some plugins can generate differing output frames at different times, even if no parameters animate or no input images change. The *kOfxImageEffectFrameVarying* property set in the clip preferences action is used to flag this.

A counterexample is a solid colour generator. If it has no animating parameters, the image generated at frame 0 will be the same as the image generated at any other frame. Intelligent hosts can render a single frame and cache that for use at all other times.

On the other hand, a plugin that generates random noise at each frame and seeds its random number generator with the render time will create different images at different times. The host cannot render a single frame and cache that for use at subsequent times.

To differentiate between these two cases the *kOfxImageEffectFrameVarying* is used. If set to 1, it indicates that the effect will need to be rendered at each frame, even if no input images or parameters are varying. If set to 0, then a single frame can be rendered and used for all times if no input images or parameters vary. The default value is 0.

1.15.2 Continuously Sampled Effects

Some effects can generate images at non frame-time boundaries, even if the inputs to the effect are frame based and there is no animation.

For example a fractal cloud generator whose pattern evolves with a speed parameter can be rendered at arbitrary times, not just on frame boundaries. Hosts that are interested in sub-frame rendering can determine that the plugin supports this behaviour by examining the *kOfxImageClipPropContinuousSamples* property set in the clip preferences action. By default this is `false`.

Note: Implicitly, all retimers effects can be continuously sampled.

1.15.3 Specifying Pixel Depths

Hosts and plugins flag whether whether they can deal with input/output clips of differing pixel depths via the *kOfxImageEffectPropSupportsMultipleClipDepths* property.

If the host sets this to 0, then all effect's input and output clips will always have the same component depth, and the plugin may not remap them.

If the plugin sets this to 0, then the host will transparently map all of an effect's input and output clips to a single depth, even if the actual clips are of differing depths. In the above two cases, the common component depth chosen will be the deepest depth of any input clip mapped to a depth the plugin supports that loses the least precision. E.g.: if a plugin supported 8 bit and float images, but the deepest clip attached to it was 16 bit, the host would transparently map all clips to float.

If both the plugin and host set this to 1, then the plugin can, during the *kOfxImageEffectActionGetClipPreferences*, specify how the host is to map each clip, including the output clip. Note that this is the only case where a plugin may set the output depth.

The bitdepth must be one of:

- **kOfxBitDepthByte**
String used to label unsigned 8 bit integer samples.

- **kOfxBitDepthShort**
String used to label unsigned 16 bit integer samples.
 - **kOfxBitDepthHalf**
String used to label half-float (16 bit floating point) samples.
- Version**
Added in Version 1.4. Was in *ofxOpenGLRender.h* before.
- **kOfxBitDepthFloat**
String used to label signed 32 bit floating point samples.
 - **kOfxBitDepthNone**
String used to label unset bitdepths.

1.15.4 Specifying Pixel Components

A plugin specifies what components it is willing to accept on a clip via the *kOfxImageEffectPropSupportedComponents* on the clip's descriptor during the *kOfxImageEffectActionDescribeInContext*. This is one or more of:

- **kOfxImageComponentRGBA**
String to label images with RGBA components.
- **kOfxImageComponentRGB**
String to label images with RGB components.
- **kOfxImageComponentAlpha**
String to label images with only Alpha components.
- **kOfxImageComponentNone**
String to label something with unset components.

If an effect has multiple inputs, and each can be a range of component types, the effect may end up with component types that are incompatible for its purposes. In this case the effect will want to have the host remap the components of the inputs and to specify the components in the output.

For example, a general effect that blends two images will have two inputs, each of which may be RGBA or A. In operation, if presented with RGBA on one and A on the other, it will most likely request that the A clip be mapped to RGBA by the host and the output be RGBA as well.

In all contexts, except for the general context, mandated input clips cannot have their component types remapped, nor can the output. Optional input clips can always have their component types remapped.

In the general context, all input clips may be remapped, as can the output clip. The output clip has its default components set to be:

- RGBA if any of the inputs is RGBA
- otherwise A if the effect has any inputs

- otherwise RGBA if there are no inputs.

Note: It is a host implementation detail as to how a host actually attaches real clips to a plugin. However it must map the clip to RGBA in a manner that is transparent to the plugin. Similarly for any other component types that the plugin does not support on an input.

1.15.5 Specifying Pixel Aspect Ratios

Hosts and plugins flag whether they can deal with input/output clips of differing pixel aspect ratios via the *kOfxImageEffectPropSupportsMultipleClipPARs* property.

If the host sets this to 0, then all effect's input and output clips will always have the same pixel aspect ratio, and the plugin may not remap them.

If the plugin sets this to 0, then the host will transparently map all of an effect's input and output clips to a single pixel aspect ratio, even if the actual clips are of differing PARs.

In the above two cases, the common pixel aspect ratio chosen will be the smallest on all the inputs, as this preserves image data.

If *both* the plugin and host set this to 1, then the plugin can, during *kOfxImageEffectActionGetClipPreferences*, specify how the host is to map each clip, including the output clip.

1.15.6 Specifying Fielding

The *kOfxImageEffectPropSettableFielding* host property indicates if a plugin is able to change the fielding of the output clip from the default.

The default value of the output clip's fielding is host dependent, but in general,

- if any of the input clips are fielded, so will the output clip
- the output clip may be fielded regardless of the input clips (for example, in a fielded project).

If the host allows a plugin to specify the fielding of the output clip, then a plugin may do so during the *kOfxImageEffectActionGetClipPreferences* by setting the property *kOfxImageClipPropFieldOrder* in the out args argument of the action. For example a defielding plugin will want to indicate that the output is frame based rather than fielded.

1.15.7 Specifying Frame Rates

The *kOfxImageEffectPropSettableFrameRate* host property indicates if a plugin is able to change the frame rate of the output clip from the default.

The default value of the output clip's frame rate is host dependent, but in general, it will be based on the input clips' frame rates.

If the host allows a plugin to specify the frame rate of the output clip, then a plugin may do so during the *kOfxImageEffectActionGetClipPreferences*. For example a deinterlace plugin that separates both fields from fielded footage will want to double the frame rate of the output clip.

If a plugin changes the frame rate, it is effectively changing the number of frames in the output clip. If our hypothetical deinterlace plugin doubles the frame rate of the output clip, it will be doubling the number of frames in that clip. The timing diagram below should help, showing how our fielded input has been turned into twice the number of frames on output.

FIELDID	SOURCE	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5
DEINTERLACED	OUTPUT	0	1	2	3	4	5	6	7	8	9	

The mapping of the number of output frames is as follows:

$$nFrames' = nFrames * FPS' / FPS$$

- `nFrames` is the default number of frames,
- `nFrames'` is the new number of output frames,
- `FPS` is the default frame rate,
- `FPS'` is the new frame rate specified by a plugin.

1.15.8 Specifying Premultiplication

All clips have a premultiplication state (see [this](#) for a nice explanation). An effect cannot map the premultiplication state of the input clips, but it can specify the premultiplication state of the output clip via `kOfxImageEffectPropPreMultiplication`, setting that to `kOfxImagePreMultiplied` or `kOfxImageUnPreMultiplied`.

The output's default premultiplication state is...

- premultiplied if any of the inputs are premultiplied
- otherwise unpremultiplied if any of the inputs are unpremultiplied
- otherwise opaque

1.16 Actions Passed to An Image Effect

Actions passed to an OFX Image Effect's plug-in main entry point are from two categories...

- actions that could potentially be issued to any kind of plug in, not just image effects, known as generic actions, found in `ofxCore.h`
- actions that are only applicable purely to image effects, found in `ofxImageEffect.h`

For generic actions, the `handle` passed to the main entry point will depend on the API being implemented, for all generic actions passed to an OFX Image Effect plug-in, it will nearly always be an `OfxImageEffectHandle`.

Because interacts are a special case, they are dealt with in a separate chapter, this chapter will deal with actions issued to an image effect plug-ins main entry point.

`kOfxActionLoad`

This action is the first action passed to a plug-in after the binary containing the plug-in has been loaded. It is there to allow a plug-in to create any global data structures it may need and is also when the plug-in should fetch suites from the host.

The `handle`, `inArgs` and `outArgs` arguments to the `mainEntry` are redundant and should be set to `NULL`.

Pre

- The plugin's `OfxPlugin::setHost` function has been called

Post

This action will not be called again while the binary containing the plug-in remains loaded.

Returns

- *kOfxStatOK*, the action was trapped and all was well,
- *kOfxStatReplyDefault*, the action was ignored,
- *kOfxStatFailed*, the load action failed, no further actions will be passed to the plug-in. Interpret if possible *kOfxStatFailed* as plug-in indicating it does not want to load Do not create an entry in the host's UI for plug-in then.

Plug-in also has the option to return 0 for *OfxGetNumberOfPlugins* or *kOfxStatFailed* if host supports *OfxSetHost* in which case *kOfxActionLoad* will never be called.

- *kOfxStatErrFatal*, fatal error in the plug-in.

kOfxActionUnload

This action is the last action passed to the plug-in before the binary containing the plug-in is unloaded. It is there to allow a plug-in to destroy any global data structures it may have created.

The handle, *inArgs* and *outArgs* arguments to the main entry are redundant and should be set to NULL.

Pre

- the *kOfxActionLoad* action has been called
- all instances of a plugin have been destroyed

Post

- No other actions will be called.

Returns

- *kOfxStatOK*, the action was trapped all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*, in which case we the program will be forced to quit

kOfxActionDescribe

The *kOfxActionDescribe* is the second action passed to a plug-in. It is where a plugin defines how it behaves and the resources it needs to function.

Note that the handle passed in acts as a descriptor for, rather than an instance of the plugin. The handle is global and unique. The plug-in is at liberty to cache the handle away for future reference until the plug-in is unloaded.

Most importantly, the effect must set what image effect contexts it is capable of working in.

This action *must* be trapped, it is not optional.

Parameters

- **handle** – handle to the plug-in descriptor, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionLoad* has been called

Post

- *kOfxActionDescribe* will not be called again, unless it fails and returns one of the error codes where the host is allowed to attempt the action again

- the handle argument, being the global plug-in description handle, is a valid handle from the end of a successful describe action until the end of the *kOfxActionUnload* action (ie: the plug-in can cache it away without worrying about it changing between actions).
- *kOfxImageEffectActionDescribeInContext* will be called once for each context that the host and plug-in mutually support. If a plug-in does not report to support any context supported by host, host should not enable the plug-in.

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatErrMissingHostFeature*, in which the plugin will be unloaded and ignored, plugin may post message
- *kOfxStatErrMemory*, in which case describe may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxActionCreateInstance

This action is the first action passed to a plug-in's instance after its creation. It is there to allow a plugin to create any per-instance data structures it may need.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionDescribe* has been called
- the instance is fully constructed, with all objects requested in the describe actions (eg, parameters and clips) have been constructed and have had their initial values set. This means that if the values are being loaded from an old setup, that load should have taken place before the create instance action is called.

Post

- the instance pointer will be valid until the *kOfxActionDestroyInstance* action is passed to the plug-in with the same instance handle

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored, but all was well anyway
- *kOfxStatErrFatal*
- *kOfxStatErrMemory*, in which case this may be called again after a memory purge
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message if possible and the host should destroy the instance handle and not attempt to proceed further

kOfxActionDestroyInstance

This action is the last passed to a plug-in's instance before its destruction. It is there to allow a plugin to destroy any per-instance data structures it may have created.

- *kOfxStatOK*, the action was trapped and all was well,
- *kOfxStatReplyDefault*, the action was ignored as the effect had nothing to do,
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the handle,
- the instance has not had any of its members destroyed yet,

Post

- the instance pointer is no longer valid and any operation on it will be undefined

Returns

To some extent, what is returned is moot, a bit like throwing an exception in a C++ destructor, so the host should continue destruction of the instance regardless.

kOfxActionBeginInstanceChanged

The *kOfxActionBeginInstanceChanged* and *kOfxActionEndInstanceChanged* actions are used to bracket all *kOfxActionInstanceChanged* actions, whether a single change or multiple changes. Some changes to a plugin instance can be grouped logically (eg: a ‘reset all’ button resetting all the instance’s parameters), the begin/end instance changed actions allow a plugin to respond appropriately to a large set of changes. For example, a plugin that maintains a complex internal state can delay any changes to that state until all parameter changes have completed.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropChangeReason* what triggered the change, which will be one of...
 - *kOfxChangeUserEdited* - the user or host changed the instance somehow and caused a change to something, this includes undo/redos, resets and loading values from files or presets,
 - *kOfxChangePluginEdited* - the plugin itself has changed the value of the instance in some action
 - *kOfxChangeTime* - the time has changed and this has affected the value of the object because it varies over time
- **outArgs** – is redundant and is set to NULL

Post

- For *kOfxActionBeginInstanceChanged*, *kOfxActionCreateInstance* has been called on the instance handle.

- For *kOfxActionEndInstanceChanged* , *kOfxActionBeginInstanceChanged* has been called on the instance handle.
- *kOfxActionCreateInstance* has been called on the instance handle.

Post

- For *kOfxActionBeginInstanceChanged*, *kOfxActionInstanceChanged* will be called at least once on the instance handle.
- *kOfxActionEndInstanceChanged* will be called on the instance handle.

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionEndInstanceChanged

Action called after the end of a set of *kOfxActionEndInstanceChanged* actions, used with *kOfxActionBeginInstanceChanged* to bracket a grouped set of changes, see *kOfxActionBeginInstanceChanged*.

kOfxActionInstanceChanged

This action signals that something has changed in a plugin's instance, either by user action, the host or the plugin itself. All change actions are bracketed by a pair of *kOfxActionBeginInstanceChanged* and *kOfxActionEndInstanceChanged* actions. The *inArgs* property set is used to determine what was the thing inside the instance that was changed.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropType* The type of the thing that changed which will be one of..
 - *kOfxTypeParameter* Indicating a parameter's value has changed in some way
 - *kOfxTypeClip* A clip to an image effect has changed in some way (for Image Effect Plugins only)
 - *kOfxPropName* the name of the thing that was changed in the instance
 - *kOfxPropChangeReason* what triggered the change, which will be one of..
 - *kOfxChangeUserEdited* - the user or host changed the instance somehow and caused a change to something, this includes undo/redos, resets and loading values from files or presets,
 - *kOfxChangePluginEdited* - the plugin itself has changed the value of the instance in some action
 - *kOfxChangeTime* - the time has changed and this has affected the value of the object because it varies over time
 - *kOfxPropTime*
 - the effect time at which the change occurred (for Image Effect Plugins only)

- *kOfxImageEffectPropRenderScale*
- the render scale currently being applied to any image fetched from a clip (for Image Effect Plugins only)
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- *kOfxActionBeginInstanceChanged* has been called on the instance handle.

Post

- *kOfxActionEndInstanceChanged* will be called on the instance handle.

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionPurgeCaches

This action is an action that may be passed to a plug-in instance from time to time in low memory situations. Instances receiving this action should destroy any data structures they may have and release the associated memory, they can later reconstruct this from the effect's parameter set and associated information.

For Image Effects, it is generally a bad idea to call this after each render, but rather it should be called after *kOfxImageEffectActionEndSequenceRender*. Some effects, typically those flagged with the *kOfxImageEffectInstancePropSequentialRender* property, may need to cache information from previously rendered frames to function correctly, or have data structures that are expensive to reconstruct at each frame (eg: a particle system). Ideally, such effect should free such structures during the *kOfxImageEffectActionEndSequenceRender* action.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionSyncPrivateData

This action is called when a plugin should synchronise any private data structures to its parameter set. This generally occurs when an effect is about to be saved or copied, but it could occur in other situations as well.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Post

- Any private state data can be reconstructed from the parameter set,

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionBeginInstanceEdit

This is called when an instance is *first* actively edited by a user, ie: and interface is open and parameter values and input clips can be modified. It is there so that effects can create private user interface structures when necessary. Note that some hosts can have multiple editors open on the same effect instance simulateously.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Post

- *kOfxActionEndInstanceEdit* will be called when the last editor is closed on the instance

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionEndInstanceEdit

This is called when the *last* user interface on an instance closed. It is there so that effects can destroy private user interface structures when necessary. Note that some hosts can have multiple editors open on the same effect instance simulateously, this will only be called when the last of those editors are closed.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionBeginInstanceEdit* has been called on the instance handle,

Post

- no user interface is open on the instance

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxImageEffectActionDescribeInContext

This action is unique to OFX Image Effect plug-ins. Because a plugin is able to exhibit different behaviour depending on the context of use, each separate context will need to be described individually. It is within this action that image effects describe which parameters and input clips it requires.

This action will be called multiple times, one for each of the contexts the plugin says it is capable of implementing. If a host does not support a certain context, then it need not call *kOfxImageEffectActionDescribeInContext* for that context.

This action *must* be trapped, it is not optional.

Parameters

- **handle** – handle to the context descriptor, cast to an *OfxImageEffectHandle* this may or may not be the same as passed to *kOfxActionDescribe*
- **inArgs** – has the following property:
 - *kOfxImageEffectPropContext* the context being described
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionDescribe* has been called on the descriptor handle,
- *kOfxActionCreateInstance* has not been called

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatErrMissingHostFeature*, in which the context will be ignored by the host, the plugin may post a message
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetRegionOfDefinition

The region of definition for an image effect is the rectangular section of the 2D image plane that it is capable of filling, given the state of its input clips and parameters. This action is used to calculate the RoD for a plugin instance at a given frame. For more details on regions of definition see Image Effect Architectures.

Note that hosts that have constant sized imagery need not call this action, only hosts that allow image sizes to vary need call this.

If the effect did not trap this, it means the host should use the default RoD instead, which depends on the context. This is...

- generator context - defaults to the project window,
- filter and paint contexts - defaults to the RoD of the ‘Source’ input clip at the given time,
- transition context - defaults to the union of the RoDs of the ‘SourceFrom’ and ‘SourceTo’ input clips at the given time,
- general context - defaults to the union of the RoDs of all the non optional input clips and the ‘Source’ input clip (if it exists and it is connected) at the given time, if none exist, then it is the project window
- retimer context - defaults to the union of the RoD of the ‘Source’ input clip at the frame directly preceding the value of the ‘SourceTime’ double parameter and the frame directly after it

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the effect time for which a region of definition is being requested
 - *kOfxImageEffectPropRenderScale* the render scale that should be used in any calculations in this action
- **outArgs** – has the following property which the plug-in may set
 - *kOfxImageEffectPropRegionOfDefinition* the calculated region of definition, initially set by the host to the default RoD (see below), in Canonical Coordinates.

Returns

- *kOfxStatOK* the action was trapped and the RoD was set in the outArgs property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetRegionsOfInterest

This action allows a host to ask an effect, given a region I want to render, what region do you need from each of your input clips. In that way, depending on the host architecture, a host can fetch the minimal amount of the image needed as input. Note that there is a region of interest to be set in outArgs for each input clip that exists on the effect. For more details see Image Effect Architectures”.

The default RoI is simply the value passed in on the *kOfxImageEffectPropRegionOfInterest* inArgs property set. All the RoIs in the outArgs property set must be initialised to this value before the action is called.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the effect time for which a region of definition is being requested
 - *kOfxImageEffectPropRenderScale* the render scale that should be used in any calculations in this action
 - *kOfxImageEffectPropRegionOfInterest* the region to be rendered in the output image, in Canonical Coordinates.
- **outArgs** – has a set of 4 dimensional double properties, one for each of the input clips to the effect. The properties are each named *OfxImageClipPropRoI_* with the clip name post pended, for example *OfxImageClipPropRoI_Source*. These are initialised to the default RoI.

Returns

- *kOfxStatOK*, the action was trapped and at least one RoI was set in the outArgs property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetFramesNeeded

This action lets the host ask the effect what frames are needed from each input clip to process a given frame. For example a temporal based degrainer may need several frames around the frame to render to do its work.

This action need only ever be called if the plugin has set the *kOfxImageEffectPropTemporalClipAccess* property on the plugin descriptor to be true. Otherwise the host assumes that the only frame needed from the inputs is the current one and this action is not called.

Note that each clip can have its required frame range specified, and that you can specify discontinuous sets of ranges for each clip, for example

```
// The effect always needs the initial frame of the source as well as the previous,
↳and current frame
double rangeSource[4];

// required ranges on the source
rangeSource[0] = 0; // we always need frame 0 of the source
rangeSource[1] = 0;
rangeSource[2] = currentFrame - 1; // we also need the previous and current frame,
↳on the source
rangeSource[3] = currentFrame;

gPropHost->propSetDoubleN(outArgs, "OfxImageClipPropFrameRange_Source", 4,
↳rangeSource);
```

Which sets two discontinuous `range` of frames `from the 'Source'` clip required `as input`.

The default frame range is simply the single frame, `kOfxPropTime..kOfxPropTime`, found on the `inArgs` property set. All the frame ranges in the `outArgs` property set must be initialised to this value before the action is called.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following property
 - *kOfxPropTime* the effect time for which we need to calculate the frames needed on input
 - `outArgs` has a set of properties, one for each input clip, named `OfxImageClipPropFrameRange_` with the name of the clip post-pended. For example `OfxImageClipPropFrameRange_Source`. All these properties are multi-dimensional doubles, with the dimension is a multiple of two. Each pair of values indicates a continuous range of frames that is needed on the given input. They are all initialised to the default value.

Returns

- *kOfxStatOK*, the action was trapped and at least one frame range in the `outArgs` property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionIsIdentity

Sometimes an effect can pass through an input unprocessed, for example a blur effect with a blur size of 0. This action can be called by a host before it attempts to render an effect to determine if it can simply copy input directly to output without having to call the render action on the effect.

If the effect does not need to process any pixels, it should set the value of the *kOfxPropName* to the clip that the host should use as the output instead, and the *kOfxPropTime* property on `outArgs` to be the time at which the frame should be fetched from a clip.

The default action is to call the render action on the effect.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the time at which to test for identity
 - *kOfxImageEffectPropFieldToRender* the field to test for identity
 - *kOfxImageEffectPropRenderWindow* the window (in `\ref PixelCoordinates`) to test for identity under
 - *kOfxImageEffectPropRenderScale* the scale factor being applied to the images being rendered
- **outArgs** – has the following properties which the plugin can set

- *kOfxPropName* this to the name of the clip that should be used if the effect is an identity transform, defaults to the empty string
- *kOfxPropTime* the time to use from the indicated source clip as an identity image (allowing time slips to happen), defaults to the value in *kOfxPropTime* in inArgs

Returns

- *kOfxStatOK*, the action was trapped and the effect should not have its render action called, the values in outArgs indicate what frame from which clip to use instead
- *kOfxStatReplyDefault*, the action was not trapped and the host should call the render action
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionRender

This action is where an effect gets to push pixels and turn its input clips and parameter set into an output image. This is possibly quite complicated and covered in the Rendering Image Effects chapter.

The render action *must* be trapped by the plug-in, it cannot return *kOfxStatReplyDefault*. The pixels needs be pushed I'm afraid.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the time at which to render
 - *kOfxImageEffectPropFieldToRender* the field to render
 - *kOfxImageEffectPropRenderWindow* the window (in `\ref PixelCoordinates`) to render
 - *kOfxImageEffectPropRenderScale* the scale factor being applied to the images being rendered
 - *kOfxImageEffectPropSequentialRenderStatus* whether the effect is currently being rendered in strict frame order on a single instance
 - *kOfxImageEffectPropInteractiveRenderStatus* if the render is in response to a user modifying the effect in an interactive session
 - *kOfxImageEffectPropRenderQualityDraft* if the render should be done in draft mode (e.g. for faster scrubbing)
- **outArgs** – is redundant and should be set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance
- *kOfxImageEffectActionBeginSequenceRender* has been called on the instance

Post

- *kOfxImageEffectActionEndSequenceRender* action will be called on the instance

Returns

- *kOfxStatOK*, the effect rendered happily
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge

- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionBeginSequenceRender

This action is passed to an image effect before it renders a range of frames. It is there to allow an effect to set things up for a long sequence of frames. Note that this is still called, even if only a single frame is being rendered in an interactive environment.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxImageEffectPropFrameRange* the range of frames (inclusive) that will be rendered
 - *kOfxImageEffectPropFrameStep* what is the step between frames, generally set to 1 (for full frame renders) or 0.5 (for fielded renders)
 - *kOfxPropIsInteractive* is this a single frame render due to user interaction in a GUI, or a proper full sequence render.
 - *kOfxImageEffectPropRenderScale* the scale factor to apply to images for this call
 - *kOfxImageEffectPropSequentialRenderStatus* whether the effect is currently being rendered in strict frame order on a single instance
 - *kOfxImageEffectPropInteractiveRenderStatus* if the render is in response to a user modifying the effect in an interactive session
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance

Post

- *kOfxImageEffectActionRender* action will be called at least once on the instance
- *kOfxImageEffectActionEndSequenceRender* action will be called on the instance

Returns

- *kOfxStatOK*, the action was trapped and handled cleanly by the effect,
- *kOfxStatReplyDefault*, the action was not trapped, but all is well anyway,
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge,
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message,
- *kOfxStatErrFatal*

kOfxImageEffectActionEndSequenceRender

This action is passed to an image effect after it has rendered a range of frames. It is there to allow an effect to free resources after a long sequence of frame renders. Note that this is still called, even if only a single frame is being rendered in an interactive environment.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties

- *kOfxImageEffectPropFrameRange* the range of frames (inclusive) that will be rendered
- *kOfxImageEffectPropFrameStep* what is the step between frames, generally set to 1 (for full frame renders) or 0.5 (for fielded renders),
- *kOfxPropIsInteractive*
- is this a single frame render due to user interaction in a GUI, or a proper full sequence render.
- *kOfxImageEffectPropRenderScale*
- the scale factor to apply to images for this call
- *kOfxImageEffectPropSequentialRenderStatus*
- whether the effect is currently being rendered in strict frame order on a single instance
- *kOfxImageEffectPropInteractiveRenderStatus*
- if the render is in response to a user modifying the effect in an interactive session
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance
- *kOfxImageEffectActionEndSequenceRender* action was called on the instance
- *kOfxImageEffectActionRender* action was called at least once on the instance

Returns

- *kOfxStatOK*, the action was trapped and handled cleanly by the effect,
- *kOfxStatReplyDefault*, the action was not trapped, but all is well anyway,
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge,
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message,
- *kOfxStatErrFatal*

kOfxImageEffectActionGetClipPreferences

This action allows a plugin to dynamically specify its preferences for input and output clips. Please see Image Effect Clip Preferences for more details on the behaviour. Clip preferences are constant for the duration of an effect, so this action need only be called once per clip, not once per frame.

This should be called once after creation of an instance, each time an input clip is changed, and whenever a parameter named in the *kOfxImageEffectPropClipPreferencesSlaveParam* has its value changed.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – has the following properties which the plugin can set
 - a set of char * X 1 properties, one for each of the input clips currently attached and the output clip, labelled with *OfxImageClipPropComponents_* post pended with the clip's name. This must be set to one of the component types which the host supports and the effect stated it can accept on that input

- a set of `char * X 1` properties, one for each of the input clips currently attached and the output clip, labelled with `OfxImageClipPropDepth_` post pended with the clip's name. This must be set to one of the pixel depths both the host and plugin supports
- a set of `double X 1` properties, one for each of the input clips currently attached and the output clip, labelled with `OfxImageClipPropPAR_` post pended with the clip's name. This is the pixel aspect ratio of the input and output clips. This must be set to a positive non zero double value,
- `kOfxImageEffectPropFrameRate` the frame rate of the output clip, this must be set to a positive non zero double value
- `kOfxImageClipPropFieldOrder` the fielding of the output clip
- `kOfxImageEffectPropPreMultiplication` the premultiplication of the output clip
- `kOfxImageClipPropContinuousSamples` whether the output clip can produce different images at non-frame intervals, defaults to false,
- `kOfxImageEffectFrameVarying` whether the output clip can produces different images at different times, even if all parameters and inputs are constant, defaults to false.

Returns

- `kOfxStatOK`, the action was trapped and at least one of the properties in the `outArgs` was changed from its default value
- `kOfxStatReplyDefault`, the action was not trapped and the host should use the default values
- `kOfxStatErrMemory`, in which case the action may be called again after a memory purge
- `kOfxStatFailed`, something wrong, but no error code appropriate, plugin to post message
- `kOfxStatErrFatal`

`kOfxImageEffectActionGetTimeDomain`

This action allows a host to ask an effect what range of frames it can produce images over. Only effects instantiated in the GeneralContext” can have this called on them. In all other the host is in strict control over the temporal duration of the effect.

The default is:

- the union of all the frame ranges of the non optional input clips,
- infinite if there are no non optional input clips.

Parameters

- **handle** – handle to the instance, cast to an `OfxImageEffectHandle`
- **inArgs** – is redundant and is null
- **outArgs** – has the following property
 - `kOfxImageEffectPropFrameRange` the frame range an effect can produce images for

Pre

- `kOfxActionCreateInstance` has been called on the instance
- the effect instance has been created in the general effect context

Returns

- *kOfxStatOK*, the action was trapped and the *kOfxImageEffectPropFrameRange* was set in the *outArgs* property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default value
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

1.17 Actions Passed to an Interact

This chapter describes the actions that can be issued to an interact's main entry point. Interact actions are also generic in character, they could be issued to other plug-in types rather than just Image Effects, however they are not issued directly to an effect's main entry point, they are rather issued to separate entry points which exist on specific 'interact' objects that a plugin may create.

For nearly all the actions the *handle* passed to to main entry point for an interact will be either NULL, or a value that should be cast to an *OfxInteractHandle*.

kOfxActionDescribeInteract

This action is the first action passed to an interact. It is where an interact defines how it behaves and the resources it needs to function. If not trapped, the default action is for the host to carry on as normal Note that the *handle* passed in acts as a descriptor for, rather than an instance of the interact.

Parameters

- **handle** – handle to the interact descriptor, cast to an *OfxInteractHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- The plugin has been loaded and the effect described.

Returns

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatErrMemory* in which case describe may be called again after a memory purge
- *kOfxStatFailed* something was wrong, the host should ignore the interact
- *kOfxStatErrFatal*

kOfxActionCreateInstanceInteract

This action is the first action passed to an interact instance after its creation. It is there to allow a plugin to create any per-instance data structures it may need.

Parameters

- **handle** – handle to the interact instance, cast to an *OfxInteractHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionDescribe* has been called on this interact

Post

- the instance pointer will be valid until the *kOfxActionDestroyInstance* action is passed to the plug-in with the same instance handle

Returns

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatReplyDefault* the action was ignored, but all was well anyway
- *kOfxStatErrFatal*
- *kOfxStatErrMemory* in which case this may be called again after a memory purge
- *kOfxStatFailed* in which case the host should ignore this interact

kOfxActionDestroyInstanceInteract

This action is the last passed to an interact's instance before its destruction. It is there to allow a plugin to destroy any per-instance data structures it may have created.

Parameters

- **handle** – handle to the interact instance, cast to an *OfxInteractHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the handle,
- the instance has not had any of its members destroyed yet

Post

- the instance pointer is no longer valid and any operation on it will be undefined

Returns

To some extent, what is returned is moot, a bit like throwing an exception in a C++ destructor, so the host should continue destruction of the instance regardless

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatReplyDefault* the action was ignored as the effect had nothing to do
- *kOfxStatErrFatal*
- *kOfxStatFailed* something went wrong, but no error code appropriate.

kOfxInteractActionDraw

This action is issued to an interact whenever the host needs the plugin to redraw the given interact.

The interact should either issue OpenGL calls to draw itself, or use DrawSuite calls.

If this is called via *kOfxImageEffectPluginPropOverlayInteractV2*, drawing **MUST** use DrawSuite.

If this is called via *kOfxImageEffectPluginPropOverlayInteractV1*, drawing **SHOULD** use OpenGL. Some existing plugins may use DrawSuite via *kOfxImageEffectPluginPropOverlayInteractV1* if it's supported by the host, but this is discouraged.

Note that the interact may (in the case of custom parameter GUIs) or may not (in the case of image effect overlays) be required to swap buffers, that is up to the kind of interact.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle
- the OpenGL context for this interact has been set
- the projection matrix will correspond to the interact's canonical view

Returns

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatReplyDefault* the action was ignored
- *kOfxStatErrFatal*
- *kOfxStatFailed* something went wrong, the host should ignore this interact in future

kOfxInteractActionPenMotion

This action is issued whenever the pen moves and the interact's has focus. It should be issued whether the pen is currently up or down. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
 - *kOfxInteractPropPenPosition* position of the pen in,
 - *kOfxInteractPropPenViewportPosition* position of the pen in,
 - *kOfxInteractPropPenPressure* the pressure of the pen,
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle

- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK* the host should not pass the pen motion to any other interactive object it may own that shares the same view.

Returns

- *kOfxStatOK* the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault* the action was not trapped and the host can deal with it if it wants

kOfxInteractActionPenDown

This action is issued when a pen transitions for the ‘up’ to the ‘down’ state. No openGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin,
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
 - *kOfxInteractPropPenPosition* position of the pen in
 - *kOfxInteractPropPenViewportPosition* position of the pen in
 - *kOfxInteractPropPenPressure* the pressure of the pen
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same view.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault*, the action was not trapped and the host can deal with it if it wants

kOfxInteractActionPenUp

This action is issued when a pen transitions for the ‘down’ to the ‘up’ state. No openGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin,
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
 - *kOfxInteractPropPenPosition* position of the pen in
 - *kOfxInteractPropPenViewportPosition* position of the pen in
 - *kOfxInteractPropPenPressure* the pressure of the pen
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same view.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault*, the action was not trapped and the host can deal with it if it wants

kOfxInteractActionKeyDown

This action is issued when a key on the keyboard is depressed. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxPropKeySym* single integer value representing the key that was manipulated, this may not have a UTF8 representation (eg: a return key)
 - *kOfxPropKeyString* UTF8 string representing a character key that was pressed, some keys have no UTF8 encoding, in which case this is ""
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same focus.

Returns

- *kOfxStatOK* , the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault* , the action was not trapped and the host can deal with it if it wants

kOfxInteractActionKeyUp

This action is issued when a key on the keyboard is released. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxPropKeySym* single integer value representing the key that was manipulated, this may not have a UTF8 representation (eg: a return key)
 - *kOfxPropKeyString* UTF8 string representing a character key that was pressed, some keys have no UTF8 encoding, in which case this is ""
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same focus.

Returns

- *kOfxStatOK* , the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault* , the action was not trapped and the host can deal with it if it wants

kOfxInteractActionKeyRepeat

This action is issued when a key on the keyboard is repeated. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin

- *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
- *kOfxPropKeySym* single integer value representing the key that was manipulated, this may not have a UTF8 representation (eg: a return key)
- *kOfxPropKeyString* UTF8 string representing a character key that was pressed, some keys have no UTF8 encoding, in which case this is ""
- *kOfxPropTime* the effect time at which changed occurred
- *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same focus.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault*, the action was not trapped and the host can deal with it if it wants

kOfxInteractActionGainFocus

This action is issued when an interact gains input focus. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact is being used on,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels,
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Returns

- *kOfxStatOK* the action was trapped
- *kOfxStatReplyDefault* the action was not trapped

kOfxInteractActionLoseFocus

This action is issued when an interact loses input focus. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact is being used on,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels,
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Returns

- *kOfxStatOK* the action was trapped
- *kOfxStatReplyDefault* the action was not trapped

1.18 OpenFX suites reference

This table list all suites available in the OpenFX standard

1.18.1 OfxPropertySuiteV1

The files `ofxCore.h` and `ofxProperty.h` contain the basic definitions for the property suite.

The property suite is the most basic and important suite in OFX, it is used to get and set the values of various objects defined by other suites.

A property is a named value of a specific data type, such values can be multi-dimensional, but is typically of one dimension. The name is a 'C' string literal, typically #defined in one of the various OFX header files. For example, the property labeled by the string literal "OfxPropName" is a 'C' string which holds the name of some object.

Properties are not accessed in isolation, but are grouped and accessed through a property set handle. The number and types of properties on a specific property set handle are currently strictly defined by the API that the properties are being used for. There is no scope to add new properties.

There is a naming convention for property labels and the macros #defined to them. The scheme is,

- generic properties names start with "OfxProp" + name of the property, e.g. "OfxPropTime".
- properties pertaining to a specific object with "Ofx" + object name + "Prop" + name of the property, e.g. "Ofx-ParamPropAnimates".
- the C preprocessor #define used to define the string literal is the same as the string literal, but with "k" prepended to the name. For example, #define kOfxPropLabel "OfxPropLabel"

OfxPropertySetHandle OfxPropertySetHandle Blind data type used to hold sets of properties:

```
#include "ofxCore.h"
typedef struct OfxPropertySetStruct *OfxPropertySetHandle;
```

Description

Properties are not accessed on their own, nor do they exist on their own. They are grouped and manipulated via an OfxPropertySetHandle.

Any object that has properties can be made to return it's property set handle via some call on the relevant suite. Individual properties are then manipulated with the property suite through that handle.

struct **OfxPropertySuiteV1**

The OFX suite used to access properties on OFX objects.

Public Members

OfxStatus (***propSetPointer**)(*OfxPropertySetHandle* properties, const char *property, int index, void *value)

Set a single value in a pointer property.

- **properties** handle of the thing holding the property
- **property** string labelling the property
- **index** for multidimensional properties and is dimension of the one we are setting
- **value** value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetString**)(*OfxPropertySetHandle* properties, const char *property, int index, const char *value)

Set a single value in a string property.

- **properties** handle of the thing holding the property
- **property** string labelling the property
- **index** for multidimensional properties and is dimension of the one we are setting
- **value** value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetDouble**)(*OfxPropertySetHandle* properties, const char *property, int index, double value)

Set a single value in a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` for multidimensional properties and is dimension of the one we are setting
- `value` value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetInt**)(*OfxPropertySetHandle* properties, const char *property, int index, int value)

Set a single value in an int property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` for multidimensional properties and is dimension of the one we are setting
- `value` value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetPointerN**)(*OfxPropertySetHandle* properties, const char *property, int count, void *const *value)

Set multiple values of the pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are setting in that property (ie: indices 0..count-1)
- `value` pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetStringN**)(*OfxPropertySetHandle* properties, const char *property, int count, const char *const *value)

Set multiple values of a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are setting in that property (ie: indices 0..count-1)
- `value` pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetDoubleN**)(*OfxPropertySetHandle* properties, const char *property, int count, const double *value)

Set multiple values of a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are setting in that property (ie: indices 0..count-1)

- value pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetIntN**)(*OfxPropertySetHandle* properties, const char *property, int count, const int *value)
Set multiple values of an int property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *count* number of values we are setting in that property (ie: indices 0..count-1)
- value pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propGetPointer**)(*OfxPropertySetHandle* properties, const char *property, int index, void **value)

Get a single value from a pointer property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *index* refers to the index of a multi-dimensional property
- value pointer the return location

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetString**)(*OfxPropertySetHandle* properties, const char *property, int index, char **value)

Get a single value of a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetDouble**)(*OfxPropertySetHandle* properties, const char *property, int index, double *value)

Get a single value of a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

See the note `ArchitectureStrings` for how to deal with strings.

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetInt**)(*OfxPropertySetHandle* properties, const char *property, int index, int *value)

Get a single value of an int property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetPointerN**)(*OfxPropertySetHandle* properties, const char *property, int count, void **value)

Get multiple values of a pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indices 0..count-1)
- `value` pointer to an array of where we will return the property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetStringN**)(*OfxPropertySetHandle* properties, const char *property, int count, char **value)

Get multiple values of a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indices 0..count-1)
- `value` pointer to an array of where we will return the property values

See the note ArchitectureStrings for how to deal with strings.

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetDoubleN**)(*OfxPropertySetHandle* properties, const char *property, int count, double **value)

Get multiple values of a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indices 0..count-1)
- `value` pointer to an array of where we will return the property values

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`
- `kOfxStatErrUnknown`
- `kOfxStatErrBadIndex`

OfxStatus (***propGetIntN**)(*OfxPropertySetHandle* properties, const char *property, int count, int *value)

Get multiple values of an int property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indices 0..count-1)
- `value` pointer to an array of where we will return the property values

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`
- `kOfxStatErrUnknown`
- `kOfxStatErrBadIndex`

OfxStatus (***propReset**)(*OfxPropertySetHandle* properties, const char *property)

Resets all dimensions of a property to its default value.

- `properties` handle of the thing holding the property
- `property` string labelling the property we are resetting

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`
- `kOfxStatErrUnknown`

OfxStatus (***propGetDimension**)(*OfxPropertySetHandle* properties, const char *property, int *count)

Gets the dimension of the property.

- properties handle of the thing holding the property
- property string labelling the property we are resetting
- count pointer to an integer where the value is returned

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*

1.18.2 OfxImageEffectSuiteV1

struct **OfxImageEffectSuiteV1**

The OFX suite for image effects.

This suite provides the functions needed by a plugin to defined and use an image effect plugin.

Public Members

OfxStatus (***getPropertySet**)(*OfxImageEffectHandle* imageEffect, *OfxPropertySetHandle* *propHandle)

Retrieves the property set for the given image effect.

- imageEffect image effect to get the property set for
- propHandle pointer to a the property set pointer, value is returned here

The property handle is for the duration of the image effect handle.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***getParamSet**)(*OfxImageEffectHandle* imageEffect, *OfiParamSetHandle* *paramSet)

Retrieves the parameter set for the given image effect.

- imageEffect image effect to get the property set for
- paramSet pointer to a the parameter set, value is returned here

The param set handle is valid for the lifetime of the image effect handle.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (*clipDefine)(*OfxImageEffectHandle* imageEffect, const char *name, *OfxPropertySetHandle* *propertySet)

Define a clip to the effect.

- pluginHandle handle passed into ‘describeInContext’ action
- name unique name of the clip to define
- propertySet property handle for the clip descriptor will be returned here

This function defines a clip to a host, the returned property set is used to describe various aspects of the clip to the host. Note that this does not create a clip instance.

Pre

- we are inside the describe in context action.

Return

OfxStatus (*clipGetHandle)(*OfxImageEffectHandle* imageEffect, const char *name, *OfxImageClipHandle* *clip, *OfxPropertySetHandle* *propertySet)

Get the property handle of the named input clip in the given instance.

- imageEffect an instance handle to the plugin
- name name of the clip, previously used in a clip define call
- clip where to return the clip
- propertySet if not NULL, the descriptor handle for a parameter’s property set will be placed here.

The propertySet will have the same value as would be returned by *OfxImageEffectSuiteV1::clipGetPropertySet*

This **return** a clip handle **for** the given instance, note that this will **not** be the same **as** the clip handle returned by clipDefine **and** will be distant to clip handles **in any** other instance of the plugin.

Not a valid call **in any** of the describe actions.

Pre

- create instance action called,
- name passed to clipDefine for this context,
- not inside describe or describe in context actions.

Post

- handle will be valid for the life time of the instance.

OfxStatus (*clipGetPropertySet)(*OfxImageClipHandle* clip, *OfxPropertySetHandle* *propHandle)

Retrieves the property set for a given clip.

- `clip` clip effect to get the property set for
- `propHandle` pointer to a the property set handle, value is returned here

The property handle is valid for the lifetime of the clip, which is generally the lifetime of the instance.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***clipGetImage**)(*OfxImageClipHandle* clip, *OfxTime* time, const *OfxRectD* *region, *OfxPropertySetHandle* *imageHandle)

Get a handle for an image in a clip at the indicated time and indicated region.

- `clip` clip to extract the image from
- `time` time to fetch the image at
- `region` region to fetch the image from (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- `imageHandle` property set containing the image's data

An image is fetched from a clip at the indicated time for the given region and returned in the `imageHandle`.

If the *region* parameter is not set to NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped to the clip's Region of Definition.

If `clipGetImage` is called twice with the same parameters, then two separate image handles will be returned, each of which must be released. The underlying implementation could share image data pointers and use reference counting to maintain them.

Pre

- `clip` was returned by `clipGetHandle`

Post

- image handle is only valid for the duration of the action `clipGetImage` is called in
- image handle to be disposed of by `clipReleaseImage` before the action returns

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time and/or region, the plugin should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - the host had not enough memory to complete the operation, plugin should abort whatever it was doing.

OfxStatus (***clipReleaseImage**)(*OfxPropertySetHandle* imageHandle)

Releases the image handle previously returned by clipGetImage.

Pre

- imageHandle was returned by clipGetImage

Post

- all operations on imageHandle will be invalid

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatErrBadHandle* - the image handle was invalid,

OfxStatus (***clipGetRegionOfDefinition**)(*OfxImageClipHandle* clip, *OfxTime* time, *OfxRectD* *bounds)

Returns the spatial region of definition of the clip at the given time.

- clipHandle clip to extract the image from
- time time to fetch the image at
- region region to fetch the image from (optional, set to NULL to get a ‘default’ region) this is in the CanonicalCoordinates.
- imageHandle handle where the image is returned

An image is fetched from a clip at the indicated time for the given region and returned in the imageHandle.

If the *region* parameter is not set to NULL, then it will be clipped to the clip’s Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped the clip’s Region of Definition.

Pre

- clipHandle was returned by clipGetHandle

Post

- bounds will be filled the RoD of the clip at the indicated time

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time, the plugin should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - the host had not enough memory to complete the operation, plugin should abort whatever it was doing.

int (***abort**)(*OfxImageEffectHandle* imageEffect)

Returns whether to abort processing or not.

- `imageEffect` instance of the image effect

A host may want to signal to a plugin that it should stop whatever rendering it is doing and start again. Generally this is done in interactive threads in response to users tweaking some parameter.

This function indicates whether a plugin should stop whatever processing it is doing.

Return

- 0 if the effect should continue whatever processing it is doing
- 1 if the effect should abort whatever processing it is doing

OfxStatus (***imageMemoryAlloc**)(*OfxImageEffectHandle* instanceHandle, size_t nBytes, *OfxImageMemoryHandle* *memoryHandle)

Allocate memory from the host's image memory pool.

- `instanceHandle` effect instance to associate with this memory allocation, may be NULL.
- `nBytes` number of bytes to allocate
- `memoryHandle` pointer to the memory handle where a return value is placed

Memory handles allocated by this should be freed by *OfxImageEffectSuiteV1::imageMemoryFree*. To access the memory behind the handle you need to call *OfxImageEffectSuiteV1::imageMemoryLock*.

See `ImageEffectsMemoryAllocation`.

Return

- `kOfxStatOK` if all went well, a valid memory handle is placed in *memoryHandle*
- `kOfxStatErrBadHandle` if `instanceHandle` is not valid, `memoryHandle` is set to NULL
- `kOfxStatErrMemory` if there was not enough memory to satisfy the call, `memoryHandle` is set to NULL

OfxStatus (***imageMemoryFree**)(*OfxImageMemoryHandle* memoryHandle)

Frees a memory handle and associated memory.

- `memoryHandle` memory handle returned by `imageMemoryAlloc`

This function frees a memory handle and associated memory that was previously allocated via *OfxImageEffectSuiteV1::imageMemoryAlloc*

If there are outstanding locks, these are ignored and the handle and memory are freed anyway.

See `ImageEffectsMemoryAllocation`.

Return

- `kOfxStatOK` if the memory was cleanly deleted
- `kOfxStatErrBadHandle` if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*

OfxStatus (***imageMemoryLock**)(*OfxImageMemoryHandle* memoryHandle, void **returnedPtr)

Lock the memory associated with a memory handle and make it available for use.

- `memoryHandle` memory handle returned by `imageMemoryAlloc`
- `returnedPtr` where to the pointer to the locked memory

This function locks them memory associated with a memory handle and returns a pointer to it. The memory will be 16 byte aligned, to allow use of vector operations.

Note that memory locks and unlocks nest.

After the first lock call, the contents of the memory pointer to by `returnedPtr` is undefined. All subsequent calls to lock will return memory with the same contents as the previous call.

Also, if unlocked, then relocked, the memory associated with a memory handle may be at a different address.

See also `OfxImageEffectSuiteV1::imageMemoryUnlock` and `ImageEffectsMemoryAllocation`.

Return

- `kOfxStatOK` if the memory was locked, a pointer is placed in `returnedPtr`
- `kOfxStatErrBadHandle` if the value of `memoryHandle` was not a valid pointer returned by `OfxImageEffectSuiteV1::imageMemoryAlloc`, null is placed in `*returnedPtr`
- `kOfxStatErrMemory` if there was not enough memory to satisfy the call, `*returnedPtr` is set to `NULL`

`OfxStatus (*imageMemoryUnlock)(OfxImageMemoryHandle memoryHandle)`

Unlock allocated image data.

- `allocatedData` pointer to memory previously returned by `OfxImageEffectSuiteV1::imageAlloc`

This function unlocks a previously locked memory handle. Once completely unlocked, memory associated with a `memoryHandle` is no longer available for use. Attempting to use it results in undefined behaviour.

Note that locks and unlocks nest, and to fully unlock memory you need to match the count of locks placed upon it.

Also note, if you unlock a completely unlocked handle, it has no effect (ie: the lock count can't be negative).

If unlocked, then relocked, the memory associated with a memory handle may be at a different address, however the contents will remain the same.

See also `OfxImageEffectSuiteV1::imageMemoryLock` and `ImageEffectsMemoryAllocation`.

Return

- `kOfxStatOK` if the memory was unlocked cleanly,
- `kOfxStatErrBadHandle` if the value of `memoryHandle` was not a valid pointer returned by `OfxImageEffectSuiteV1::imageMemoryAlloc`, null is placed in `*returnedPtr`

1.18.3 OfxProgressSuiteV1

struct **OfxProgressSuiteV1**

A suite that provides progress feedback from a plugin to an application.

A plugin instance can initiate, update and close a progress indicator with this suite.

This is an optional suite in the Image Effect API.

API V1.4: Amends the documentation of progress suite V1 so that it is expected that it can be raised in a modal manner and have a “cancel” button when invoked in instanceChanged. Plugins that perform analysis post an appropriate message, raise the progress monitor in a modal manner and should poll to see if processing has been aborted. Any cancellation should be handled gracefully by the plugin (eg: reset analysis parameters to default values), clear allocated memory...

Many hosts already operate as described above. kOfxStatReplyNo should be returned to the plugin during progressUpdate when the user presses cancel.

Suite V2: Adds an ID that can be looked up for internationalisation and so on. When a new version is introduced, because plug-ins need to support old versions, and plug-in’s new releases are not necessary in synch with hosts (or users don’t immediately update), best practice is to support the 2 suite versions. That is, the plugin should check if V2 exists; if not then check if V1 exists. This way a graceful transition is guaranteed. So plugin should fetchSuite passing 2, (OfxProgressSuiteV2*) fetchSuite(mHost->mHost->host, kOfxProgressSuite,2); and if no success pass (OfxProgressSuiteV1*) fetchSuite(mHost->mHost->host, kOfxProgressSuite,1);

Public Members

OfxStatus (***progressStart**)(void *effectInstance, const char *label)

Initiate a progress bar display.

Call this to initiate the display of a progress bar.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.
- *label* a text label to display in any message portion of the progress object’s user interface. A UTF8 string.

Pre

- There is no currently ongoing progress display for this instance.

Return

- *kOfxStatOK* - the handle is now valid for use
- *kOfxStatFailed* - the progress object failed for some reason
- *kOfxStatErrBadHandle* - effectInstance was invalid

OfxStatus (***progressUpdate**)(void *effectInstance, double progress)

Indicate how much of the processing task has been completed and reports on any abort status.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.

- **progress** a number between 0.0 and 1.0 indicating what proportion of the current task has been processed.

Return

- *kOfxStatOK* - the progress object was successfully updated and the task should continue
- *kOfxStatReplyNo* - the progress object was successfully updated and the task should abort
- *kOfxStatErrBadHandle* - the progress handle was invalid,

OfxStatus (***progressEnd**)(void *effectInstance)

Signal that we are finished with the progress meter.

Call this when you are done with the progress meter and no longer need it displayed.

- **effectInstance** the instance of the plugin this progress bar is associated with. It cannot be NULL.

Post

- you can no longer call `progressUpdate` on the instance

Return

- *kOfxStatOK* - the progress object was successfully closed
- *kOfxStatErrBadHandle* - the progress handle was invalid,

1.18.4 OfxTimeLineSuiteV1

struct **OfxImageEffectSuiteV1**

The OFX suite for image effects.

This suite provides the functions needed by a plugin to defined and use an image effect plugin.

Public Members

OfxStatus (***getPropertySet**)(*OfxImageEffectHandle* imageEffect, *OfxPropertySetHandle* *propHandle)

Retrieves the property set for the given image effect.

- **imageEffect** image effect to get the property set for
- **propHandle** pointer to a the property set pointer, value is returned here

The property handle is for the duration of the image effect handle.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***getParamSet**)(*OfxImageEffectHandle* imageEffect, *OfiParamSetHandle* *paramSet)

Retrieves the parameter set for the given image effect.

- imageEffect image effect to get the property set for
- paramSet pointer to a the parameter set, value is returned here

The param set handle is valid for the lifetime of the image effect handle.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***clipDefine**)(*OfxImageEffectHandle* imageEffect, const char *name, *OfxPropertySetHandle* *propertySet)

Define a clip to the effect.

- pluginHandle handle passed into 'describeInContext' action
- name unique name of the clip to define
- propertySet property handle for the clip descriptor will be returned here

This function defines a clip to a host, the returned property set is used to describe various aspects of the clip to the host. Note that this does not create a clip instance.

Pre

- we are inside the describe in context action.

Return

OfxStatus (***clipGetHandle**)(*OfxImageEffectHandle* imageEffect, const char *name, *OfxImageClipHandle* *clip, *OfxPropertySetHandle* *propertySet)

Get the property handle of the named input clip in the given instance.

- imageEffect an instance handle to the plugin
- name name of the clip, previously used in a clip define call
- clip where to return the clip
- propertySet if not NULL, the descriptor handle for a parameter's property set will be placed here.

The propertySet will have the same value as would be returned by *OfxImageEffectSuiteV1::clipGetPropertySet*

This **return** a clip handle **for** the given instance, note that this will **not** be the same **as** the clip handle returned by clipDefine **and** will be distant to clip handles **in any** other instance of the plugin.

(continues on next page)

(continued from previous page)

Not a valid call **in any** of the describe actions.

Pre

- create instance action called,
- *name* passed to clipDefine for this context,
- not inside describe or describe in context actions.

Post

- handle will be valid for the life time of the instance.

OfxStatus (*clipGetPropertySet)(*OfxImageClipHandle* clip, *OfxPropertySetHandle* *propHandle)

Retrieves the property set for a given clip.

- clip clip effect to get the property set for
- propHandle pointer to a the property set handle, value is returned here

The property handle is valid for the lifetime of the clip, which is generally the lifetime of the instance.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (*clipGetImage)(*OfxImageClipHandle* clip, *OfxTime* time, const *OfxRectD* *region, *OfxPropertySetHandle* *imageHandle)

Get a handle for an image in a clip at the indicated time and indicated region.

- clip clip to extract the image from
- time time to fetch the image at
- region region to fetch the image from (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- imageHandle property set containing the image's data

An image is fetched from a clip at the indicated time for the given region and returned in the imageHandle.

If the *region* parameter is not set to NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped the clip's Region of Definition.

If clipGetImage is called twice with the same parameters, then two separate image handles will be returned, each of which must be release. The underlying implementation could share image data pointers and use reference counting to maintain them.

Pre

- clip was returned by clipGetHandle

Post

- image handle is only valid for the duration of the action clipGetImage is called in
- image handle to be disposed of by clipReleaseImage before the action returns

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time and/or region, the plugin should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - the host had not enough memory to complete the operation, plugin should abort whatever it was doing.

OfxStatus (*clipReleaseImage)(*OfxPropertySetHandle* imageHandle)

Releases the image handle previously returned by clipGetImage.

Pre

- imageHandle was returned by clipGetImage

Post

- all operations on imageHandle will be invalid

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatErrBadHandle* - the image handle was invalid,

OfxStatus (*clipGetRegionOfDefinition)(*OfxImageClipHandle* clip, *OfxTime* time, *OfxRectD* *bounds)

Returns the spatial region of definition of the clip at the given time.

- clipHandle clip to extract the image from
- time time to fetch the image at
- region region to fetch the image from (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- imageHandle handle where the image is returned

An image is fetched from a clip at the indicated time for the given region and returned in the imageHandle.

If the *region* parameter is not set to NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped the clip's Region of Definition.

Pre

- clipHandle was returned by clipGetHandle

Post

- bounds will be filled the RoD of the clip at the indicated time

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time, the plugin should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - the host had not enough memory to complete the operation, plugin should abort whatever it was doing.

int (***abort**)(*OfxImageEffectHandle* imageEffect)

Returns whether to abort processing or not.

- *imageEffect* instance of the image effect

A host may want to signal to a plugin that it should stop whatever rendering it is doing and start again. Generally this is done in interactive threads in response to users tweaking some parameter.

This function indicates whether a plugin should stop whatever processing it is doing.

Return

- 0 if the effect should continue whatever processing it is doing
- 1 if the effect should abort whatever processing it is doing

OfxStatus (***imageMemoryAlloc**)(*OfxImageEffectHandle* instanceHandle, size_t nBytes, *OfxImageMemoryHandle* *memoryHandle)

Allocate memory from the host's image memory pool.

- *instanceHandle* effect instance to associate with this memory allocation, may be NULL.
- *nBytes* number of bytes to allocate
- *memoryHandle* pointer to the memory handle where a return value is placed

Memory handles allocated by this should be freed by *OfxImageEffectSuiteV1::imageMemoryFree*. To access the memory behind the handle you need to call *OfxImageEffectSuiteV1::imageMemoryLock*.

See ImageEffectsMemoryAllocation.

Return

- *kOfxStatOK* if all went well, a valid memory handle is placed in *memoryHandle*
- *kOfxStatErrBadHandle* if *instanceHandle* is not valid, *memoryHandle* is set to NULL
- *kOfxStatErrMemory* if there was not enough memory to satisfy the call, *memoryHandle* is set to NULL

OfxStatus (***imageMemoryFree**)(*OfxImageMemoryHandle* memoryHandle)

Frees a memory handle and associated memory.

- *memoryHandle* memory handle returned by *imageMemoryAlloc*

This function frees a memory handle and associated memory that was previously allocated via *OfxImageEffectSuiteV1::imageMemoryAlloc*

If there are outstanding locks, these are ignored and the handle and memory are freed anyway.

See ImageEffectsMemoryAllocation.

Return

- kOfxStatOK if the memory was cleanly deleted
- kOfxStatErrBadHandle if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*

OfxStatus (***imageMemoryLock**)(*OfxImageMemoryHandle* memoryHandle, void **returnedPtr)

Lock the memory associated with a memory handle and make it available for use.

- *memoryHandle* memory handle returned by *imageMemoryAlloc*
- *returnedPtr* where to the pointer to the locked memory

This function locks the memory associated with a memory handle and returns a pointer to it. The memory will be 16 byte aligned, to allow use of vector operations.

Note that memory locks and unlocks nest.

After the first lock call, the contents of the memory pointer to by *returnedPtr* is undefined. All subsequent calls to lock will return memory with the same contents as the previous call.

Also, if unlocked, then relocked, the memory associated with a memory handle may be at a different address.

See also *OfxImageEffectSuiteV1::imageMemoryUnlock* and ImageEffectsMemoryAllocation.

Return

- kOfxStatOK if the memory was locked, a pointer is placed in *returnedPtr*
- kOfxStatErrBadHandle if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*, null is placed in **returnedPtr*
- kOfxStatErrMemory if there was not enough memory to satisfy the call, **returnedPtr* is set to NULL

OfxStatus (***imageMemoryUnlock**)(*OfxImageMemoryHandle* memoryHandle)

Unlock allocated image data.

- *allocatedData* pointer to memory previously returned by *OfxImageEffectSuiteV1::imageAlloc*

This function unlocks a previously locked memory handle. Once completely unlocked, memory associated with a *memoryHandle* is no longer available for use. Attempting to use it results in undefined behaviour.

Note that locks and unlocks nest, and to fully unlock memory you need to match the count of locks placed upon it.

Also note, if you unlock a completely unlocked handle, it has no effect (ie: the lock count can't be negative).

If unlocked, then relocked, the memory associated with a memory handle may be at a different address, however the contents will remain the same.

See also *OfxImageEffectSuiteV1::imageMemoryLock* and ImageEffectsMemoryAllocation.

Return

- `kOfxStatOK` if the memory was unlocked cleanly,
- `kOfxStatErrBadHandle` if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*, null is placed in **returnedPtr*

1.18.5 OfxParameterSuiteV1

struct **OfxParameterSuiteV1**

The OFX suite used to define and manipulate user visible parameters.

Keyframe Handling

These functions allow the plug-in to delete and get information about keyframes.

To set keyframes, use *paramSetValueAtTime()*.

paramGetKeyTime and *paramGetKeyIndex* use indices to refer to keyframes. Keyframes are stored by the host in increasing time order, so $\text{time}(\text{kf}[i]) < \text{time}(\text{kf}[i+1])$. Keyframe indices will change whenever keyframes are added, deleted, or moved in time, whether by the host or by the plug-in. They may vary between actions if the user changes a keyframe. The keyframe indices will not change within a single action.

OfxStatus (***paramGetNumKeys**)(*OfxParamHandle* paramHandle, unsigned int *numberOfKeys)

Returns the number of keyframes in the parameter.

- *paramHandle* parameter handle to interrogate
- *numberOfKeys* pointer to integer where the return value is placed

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Returns the number of keyframes in the parameter.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetKeyTime**)(*OfxParamHandle* paramHandle, unsigned int nthKey, *OfxTime* *time)

Returns the time of the nth key.

- *paramHandle* parameter handle to interrogate
- *nthKey* which key to ask about (0 to *paramGetNumKeys* - 1), ordered by time
- *time* pointer to *OfxTime* where the return value is placed

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

- *kOfxStatErrBadIndex* - the nthKey does not exist

OfxStatus (***paramGetKeyIndex**)(*OfxParamHandle* paramHandle, *OfxTime* time, int direction, int *index)

Finds the index of a keyframe at/before/after a specified time.

- **paramHandle** parameter handle to search
- **time** what time to search from
- **direction**
 - == 0 indicates search for a key at the indicated time (some small delta)
 - > 0 indicates search for the next key after the indicated time
 - < 0 indicates search for the previous key before the indicated time
- **index** pointer to an integer which in which the index is returned set to -1 if no key was found

Return

- *kOfxStatOK* - all was OK
- *kOfxStatFailed* - if the search failed to find a key
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramDeleteKey**)(*OfxParamHandle* paramHandle, *OfxTime* time)

Deletes a keyframe if one exists at the given time.

- **paramHandle** parameter handle to delete the key from
- **time** time at which a keyframe is

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrBadIndex* - no key at the given time

OfxStatus (***paramDeleteAllKeys**)(*OfxParamHandle* paramHandle)

Deletes all keyframes from a parameter.

- **paramHandle** parameter handle to delete the keys from
- **name** parameter to delete the keyframes from is

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

Public Members

OfxStatus (***paramDefine**)(*OfiParamSetHandle* paramSet, const char *paramType, const char *name, *OfxPropertySetHandle* *propertySet)

Defines a new parameter of the given type in a describe action.

- paramSet handle to the parameter set descriptor that will hold this parameter
- paramType type of the parameter to create, one of the *kOfiParamType** #defines
- name unique name of the parameter
- propertySet if not null, a pointer to the parameter descriptor's property set will be placed here.

This function defines a parameter in a parameter set and returns a property set which is used to describe that parameter.

This function does not actually create a parameter, it only says that one should exist in any subsequent instances. To fetch an parameter instance *paramGetHandle* must be called on an instance.

This function can always be called in one of a plug-in's "describe" functions which defines the parameter sets common to all instances of a plugin.

Return

- *kOfxStatOK* - the parameter was created correctly
- *kOfxStatErrBadHandle* - if the plugin handle was invalid
- *kOfxStatErrExists* - if a parameter of that name exists already in this plugin
- *kOfxStatErrUnknown* - if the type is unknown
- *kOfxStatErrUnsupported* - if the type is known but unsupported

OfxStatus (***paramGetHandle**)(*OfiParamSetHandle* paramSet, const char *name, *OfiParamHandle* *param, *OfxPropertySetHandle* *propertySet)

Retrieves the handle for a parameter in a given parameter set.

- paramSet instance of the plug-in to fetch the property handle from
- name parameter to ask about
- param pointer to a param handle, the value is returned here
- propertySet if not null, a pointer to the parameter's property set will be placed here.

Parameter handles retrieved from an instance are always distinct in each instance. The parameter handle is valid for the life-time of the instance. Parameter handles in instances are distinct from parameter handles in plugins. You cannot call this in a plugin's describe function, as it needs an instance to work on.

Return

- *kOfxStatOK* - the parameter was found and returned
- *kOfxStatErrBadHandle* - if the plugin handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramSetGetPropertySet**)(*OfxParamSetHandle* paramSet, *OfxPropertySetHandle* *propHandle)

Retrieves the property set handle for the given parameter set.

- paramSet parameter set to get the property set for
- propHandle pointer to a the property set handle, value is returned here

Note: The property handle belonging to a parameter set is the same as the property handle belonging to the plugin instance.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramGetPropertySet**)(*OfxParamHandle* param, *OfxPropertySetHandle* *propHandle)

Retrieves the property set handle for the given parameter.

- param parameter to get the property set for
- propHandle pointer to a the property set handle, value is returned here

The property handle is valid for the lifetime of the parameter, which is the lifetime of the instance that owns the parameter

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramGetValue**)(*OfxParamHandle* paramHandle, ...)

Gets the current value of a parameter,.

- paramHandle parameter handle to fetch value from
- ... one or more pointers to variables of the relevant type to hold the parameter's value

This gets the current value of a parameter. The varargs ... argument needs to be pointer to C variables of the relevant type for this parameter. Note that params with multiple values (eg Colour) take multiple args here. For example...

```
OfxParamHandle myDoubleParam, *myColourParam;
ofxHost->paramGetHandle(instance, "myDoubleParam", &myDoubleParam);
double myDoubleValue;
ofxHost->paramGetValue(myDoubleParam, &myDoubleValue);
```

(continues on next page)

(continued from previous page)

```

ofxHost->paramGetHandle(instance, "myColourParam", &myColourParam);
double myR, myG, myB;
ofxHost->paramGetValue(myColourParam, &myR, &myG, &myB);

```

Note: `paramGetValue` should only be called from within a *kOfxActionInstanceChanged* or interact action and never from the render actions (which should always use `paramGetValueAtTime`).

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetValueAtTime**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Gets the value of a parameter at a specific time.

- `paramHandle` parameter handle to fetch value from
- `time` at what point in time to look up the parameter
- ... one or more pointers to variables of the relevant type to hold the parameter's value

This gets the current value of a parameter. The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteVI::paramGetValue* for notes on the varags list

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetDerivative**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Gets the derivative of a parameter at a specific time.

- `paramHandle` parameter handle to fetch value from
- `time` at what point in time to look up the parameter
- ... one or more pointers to variables of the relevant type to hold the parameter's derivative

This gets the derivative of the parameter at the indicated time.

The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteVI::paramGetValue* for notes on the varags list.

Only double and colour params can have their derivatives found.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetIntegral**)(*OfxParamHandle* paramHandle, *OfxTime* time1, *OfxTime* time2, ...)

Gets the integral of a parameter over a specific time range,.

- **paramHandle** parameter handle to fetch integral from
- **time1** where to start evaluating the integral
- **time2** where to stop evaluating the integral
- ... one or more pointers to variables of the relevant type to hold the parameter's integral

This gets the integral of the parameter over the specified time range.

The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteV1::paramGetValue* for notes on the varargs list.

Only double and colour params can be integrated.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramSetValue**)(*OfxParamHandle* paramHandle, ...)

Sets the current value of a parameter.

- **paramHandle** parameter handle to set value in
- ... one or more variables of the relevant type to hold the parameter's value

This sets the current value of a parameter. The varargs ... argument needs to be values of the relevant type for this parameter. Note that params with multiple values (eg Colour) take multiple args here. For example...

```
ofxHost->paramSetValue(instance, "myDoubleParam", double(10));
ofxHost->paramSetValue(instance, "myColourParam", double(pix.r), double(pix.
↪g), double(pix.b));
```

Note: `paramSetValue` should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramSetValueAtTime**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Keyframes the value of a parameter at a specific time.

- **paramHandle** parameter handle to set value in
- **time** at what point in time to set the keyframe

- ... one or more variables of the relevant type to hold the parameter's value

This sets a keyframe in the parameter at the indicated time to have the indicated value. The varargs ... argument needs to be values of the relevant type for this parameter. See the note on *OfxParameterSuiteV1::paramSetValue* for more detail

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Note: *paramSetValueAtTime* should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramCopy**)(*OfxParamHandle* paramTo, *OfxParamHandle* paramFrom, *OfxTime* dstOffset, const *OfxRangeD* *frameRange)

Copies one parameter to another, including any animation etc...

- *paramTo* parameter to set
- *paramFrom* parameter to copy from
- *dstOffset* temporal offset to apply to keys when writing to the *paramTo*
- *frameRange* if *paramFrom* has animation, and *frameRange* is not null, only this range of keys will be copied

This copies the value of *paramFrom* to *paramTo*, including any animation it may have. All the previous values in *paramTo* will be lost.

To choose all animation in *paramFrom* set *frameRange* to [0, 0]

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Pre

- Both parameters must be of the same type.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramEditBegin**)(*OfxParamSetHandle* paramSet, const char *name)

Used to group any parameter changes for undo/redo purposes.

- *paramSet* the parameter set in which this is happening
- *name* label to attach to any undo/redo string UTF8

If a plugin calls `paramSetValue/paramSetValueAtTime` on one or more parameters, either from custom GUI interaction or some analysis of imagery etc.. this is used to indicate the start of a set of a parameter changes that should be considered part of a single undo/redo block.

See also *OfxParameterSuiteV1::paramEditEnd*

Note: `paramEditBegin` should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the instance handle was invalid

OfxStatus (***paramEditEnd**)(*OfxParamSetHandle* paramSet)

Used to group any parameter changes for undo/redo purposes.

- `paramSet` parameter set in which this is happening

If a plugin calls `paramSetValue/paramSetValueAtTime` on one or more parameters, either from custom GUI interaction or some analysis of imagery etc.. this is used to indicate the end of a set of parameter changes that should be considered part of a single undo/redo block

See also *OfxParameterSuiteV1::paramEditBegin*

Note: `paramEditEnd` should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the instance handle was invalid

1.18.6 OfxParametricParameterSuiteV1

struct **OfxParametricParameterSuiteV1**

The OFX suite used to define and manipulate ‘parametric’ parameters.

This is an optional suite.

Parametric parameters are in effect ‘functions’ a plug-in can ask a host to arbitrarily evaluate for some value ‘x’. A classic use case would be for constructing look-up tables, a plug-in would ask the host to evaluate one at multiple values from 0 to 1 and use that to fill an array.

A host would probably represent this to a user as a cubic curve in a standard curve editor interface, or possibly through scripting. The user would then use this to define the ‘shape’ of the parameter.

The evaluation of such params is not the same as animation, they are returning values based on some arbitrary argument orthogonal to time, so to evaluate such a param, you need to pass a parametric position and time.

Often, you would want such a parametric parameter to be multi-dimensional, for example, a colour look-up table might want three values, one for red, green and blue. Rather than declare three separate parametric parameters, it would be better to have one such parameter with multiple values in it.

The major complication with these parameters is how to allow a plug-in to set values, and defaults. The default default value of a parametric curve is to be an identity lookup. If a plugin wishes to set a different default value for a curve, it can use the suite to set key/value pairs on the *descriptor* of the param. When a new instance is made, it will have these curve values as a default.

Public Members

OfxStatus (***parametricParamGetValue**)(*OfxParamHandle* param, int curveIndex, *OfxTime* time, double parametricPosition, double *returnValue)

Evaluates a parametric parameter.

- param handle to the parametric parameter
- curveIndex which dimension to evaluate
- time the time to evaluate to the parametric param at
- parametricPosition the position to evaluate the parametric param at
- returnValue pointer to a double where a value is returned

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrBadIndex* - the curve index was invalid

OfxStatus (***parametricParamGetNControlPoints**)(*OfxParamHandle* param, int curveIndex, double time, int *returnValue)

Returns the number of control points in the parametric param.

- param handle to the parametric parameter
- curveIndex which dimension to check
- time the time to check
- returnValue pointer to an integer where the value is returned.

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrBadIndex* - the curve index was invalid

OfxStatus (***parametricParamGetNthControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, int nthCtl, double *key, double *value)

Returns the key/value pair of the nth control point.

- param handle to the parametric parameter
- curveIndex which dimension to check
- time the time to check
- nthCtl the nth control point to get the value of
- key pointer to a double where the key will be returned
- value pointer to a double where the value will be returned

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***parametricParamSetNthControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, int nthCtl, double key, double value, bool addAnimationKey)

Modifies an existing control point on a curve.

- param handle to the parametric parameter
- curveIndex which dimension to set
- time the time to set the value at
- nthCtl the control point to modify
- key key of the control point
- value value of the control point
- addAnimationKey if the param is an animatable, setting this to true will force an animation keyframe to be set as well as a curve key, otherwise if false, a key will only be added if the curve is already animating.

This modifies an existing control point. Note that by changing key, the order of the control point may be modified (as you may move it before or after another point). So be careful when iterating over a curves control points and you change a key.

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***parametricParamAddControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, double key, double value, bool addAnimationKey)

Adds a control point to the curve.

- `param` handle to the parametric parameter
- `curveIndex` which dimension to set
- `time` the time to set the value at
- `key` key of the control point
- `value` value of the control point
- `addAnimationKey` if the `param` is an animatable, setting this to true will force an animation keyframe to be set as well as a curve key, otherwise if false, a key will only be added if the curve is already animating.

This will add a new control point to the given dimension of a parametric parameter. If a key exists sufficiently close to 'key', then it will be set to the indicated control point.

Return

- `kOfxStatOK` - all was fine
- `kOfxStatErrBadHandle` - if the parameter handle was invalid
- `kOfxStatErrUnknown` - if the type is unknown

OfxStatus (***parametricParamDeleteControlPoint**)(*OfxParamHandle* param, int curveIndex, int nthCtl)

Deletes the nth control point from a parametric param.

- `param` handle to the parametric parameter
- `curveIndex` which dimension to delete
- `nthCtl` the control point to delete

OfxStatus (***parametricParamDeleteAllControlPoints**)(*OfxParamHandle* param, int curveIndex)

Delete all curve control points on the given param.

- `param` handle to the parametric parameter
- `curveIndex` which dimension to clear

1.18.7 OfxMemorySuiteV1

struct **OfxMemorySuiteV1**

The OFX suite that implements general purpose memory management.

Use this suite for ordinary memory management functions, where you would normally use malloc/free or new/delete on ordinary objects.

For images, you should use the memory allocation functions in the image effect suite, as many hosts have specific image memory pools.

Note: C++ plugin developers will need to redefine new and delete as skins ontop of this suite.

Public Members

OfxStatus (***memoryAlloc**)(void *handle, size_t nBytes, void **allocatedData)

Allocate memory.

- *handle* - effect instance to associate with this memory allocation, or NULL.
- *nBytes* number of bytes to allocate
- *allocatedData* pointer to the return value. Allocated memory will be alligned for any use.

This function has the host allocate memory using its own memory resources and returns that to the plugin.

Return

- *kOfxStatOK* the memory was sucessfully allocated
- *kOfxStatErrMemory* the request could not be met and no memory was allocated

OfxStatus (***memoryFree**)(void *allocatedData)

Frees memory.

- *allocatedData* pointer to memory previously returned by *OfxMemorySuiteV1::memoryAlloc*

This function frees any memory that was previously allocated via *OfxMemorySuiteV1::memoryAlloc*.

Return

- *kOfxStatOK* the memory was sucessfully freed
- *kOfxStatErrBadHandle* *allocatedData* was not a valid pointer returned by *OfxMemorySuiteV1::memoryAlloc*

1.18.8 OfxMultiThreadSuiteV1

struct **OfxMultiThreadSuiteV1**

OFX suite that provides simple SMP style multi-processing.

Public Members

OfxStatus (***multiThread**)(OfxThreadFunctionV1 func, unsigned int nThreads, void *customArg)

Function to spawn SMP threads.

- *func* function to call in each thread.
- *nThreads* number of threads to launch

- `customArg` paramter to pass to `customArg` of `func` in each thread.

This function will spawn `nThreads` separate threads of computation (typically one per CPU) to allow something to perform symmetric multi processing. Each thread will call 'func' passing in the index of the thread and the number of threads actually launched.

`multiThread` will not return until all the spawned threads have returned. It is up to the host how it waits for all the threads to return (busy wait, blocking, whatever).

`nThreads` can be more than the value returned by `multiThreadNumCPUs`, however the threads will be limited to the number of CPUs returned by `multiThreadNumCPUs`.

This function cannot be called recursively.

Return

- `kOfxStatOK`, the function `func` has executed and returned sucessfully
- `kOfxStatFailed`, the threading function failed to launch
- `kOfxStatErrExists`, failed in an attempt to call `multiThread` recursively,

`OfxStatus (*multiThreadNumCPUs)(unsigned int *nCPUs)`

Function which indicates the number of CPUs available for SMP processing.

- `nCPUs` pointer to an integer where the result is returned

This value may be less than the actual number of CPUs on a machine, as the host may reserve other CPUs for itself.

Return

- `kOfxStatOK`, all was OK and the maximum number of threads is in `nThreads`.
- `kOfxStatFailed`, the function failed to get the number of CPUs

`OfxStatus (*multiThreadIndex)(unsigned int *threadIndex)`

Function which indicates the index of the current thread.

- `threadIndex` pointer to an integer where the result is returned

This function returns the thread index, which is the same as the `threadIndex` argument passed to the `OfxThreadFunctionV1`.

If there are no threads currently spawned, then this function will set `threadIndex` to 0

Return

- `kOfxStatOK`, all was OK and the maximum number of threads is in `nThreads`.
- `kOfxStatFailed`, the function failed to return an index

`int (*multiThreadIsSpawnedThread)(void)`

Function to enquire if the calling thread was spawned by `multiThread`.

Return

- 0 if the thread is not one spawned by `multiThread`
- 1 if the thread was spawned by `multiThread`

OfxStatus (***mutexCreate**)(*OfxMutexHandle* *mutex, int lockCount)

Create a mutex.

- `mutex` where the new handle is returned
- `count` initial lock count on the mutex. This can be negative.

Creates a new mutex with `lockCount` locks on the mutex initially set.

Return

- `kOfxStatOK` - mutex is now valid and ready to go

OfxStatus (***mutexDestroy**)(const *OfxMutexHandle* mutex)

Destroy a mutex.

Destroys a mutex initially created by `mutexCreate`.

Return

- `kOfxStatOK` - if it destroyed the mutex
- `kOfxStatErrBadHandle` - if the handle was bad

OfxStatus (***mutexLock**)(const *OfxMutexHandle* mutex)

Blocking lock on the mutex.

This tries to lock a mutex and blocks the thread it is in until the lock succeeds.

A successful lock causes the mutex's lock count to be increased by one and to block any other calls to lock the mutex until it is unlocked.

Return

- `kOfxStatOK` - if it got the lock
- `kOfxStatErrBadHandle` - if the handle was bad

OfxStatus (***mutexUnLock**)(const *OfxMutexHandle* mutex)

Unlock the mutex.

This unlocks a mutex. Unlocking a mutex decreases its lock count by one.

Return

- `kOfxStatOK` if it released the lock
- `kOfxStatErrBadHandle` if the handle was bad

OfxStatus (***mutexTryLock**)(const *OfxMutexHandle* mutex)

Non blocking attempt to lock the mutex.

This attempts to lock a mutex, if it cannot, it returns and says so, rather than blocking.

A successful lock causes the mutex's lock count to be increased by one, if the lock did not succeed, the call returns immediately and the lock count remains unchanged.

Return

- `kOfxStatOK` - if it got the lock

- `kOfxStatFailed` - if it did not get the lock
- `kOfxStatErrBadHandle` - if the handle was bad

1.18.9 OfxInteractSuiteV1

struct **OfxInteractSuiteV1**

OFX suite that allows an effect to interact with an openGL window so as to provide custom interfaces.

Public Members

OfxStatus (***interactSwapBuffers**)(*OfxInteractHandle* interactInstance)

Requests an openGL buffer swap on the interact instance.

OfxStatus (***interactRedraw**)(*OfxInteractHandle* interactInstance)

Requests a redraw of the interact instance.

OfxStatus (***interactGetPropertySet**)(*OfxInteractHandle* interactInstance, *OfxPropertySetHandle* *property)

Gets the property set handle for this interact handle.

1.18.10 OfxMessageSuiteV1

struct **OfxMessageSuiteV1**

The OFX suite that allows a plug-in to pass messages back to a user. The V2 suite extends on this in a backwards compatible manner.

Public Members

OfxStatus (***message**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a message on the host, using printf style varargs.

- `handle` effect handle (descriptor or instance) the message should be associated with, may be NULL
- `messageType` string describing the kind of message to post, one of the `kOfxMessageType*` constants
- `messageId` plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- `format` printf style format string
- ... printf style varargs list to print

Return

- `kOfxStatOK` - if the message was successfully posted

- *kOfxStatReplyYes* - if the message was of type *kOfxMessageQuestion* and the user reply yes
- *kOfxStatReplyNo* - if the message was of type *kOfxMessageQuestion* and the user reply no
- *kOfxStatFailed* - if the message could not be posted for some reason

1.18.11 OfxMessageSuiteV2

struct **OfxMessageSuiteV2**

The OFX suite that allows a plug-in to pass messages back to a user.

This extends *OfxMessageSuiteV1*, and should be considered a replacement to version 1.

Note that this suite has been extended in backwards compatible manner, so that a host can return this struct for both V1 and V2.

Public Members

OfxStatus (***message**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a transient message on the host, using printf style varargs. Same as the V1 message suite call.

- *handle* effect handle (descriptor or instance) the message should be associated with, may be null
- *messageType* string describing the kind of message to post, one of the *kOfxMessageType** constants
- *messageId* plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- *format* printf style format string
- ... printf style varargs list to print

Return

- *kOfxStatOK* - if the message was successfully posted
- *kOfxStatReplyYes* - if the message was of type *kOfxMessageQuestion* and the user reply yes
- *kOfxStatReplyNo* - if the message was of type *kOfxMessageQuestion* and the user reply no
- *kOfxStatFailed* - if the message could not be posted for some reason

OfxStatus (***setPersistentMessage**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a persistent message on an effect, using printf style varargs, and set error states. New for V2 message suite.

- *handle* effect instance handle the message should be associated with, may NOT be null,
- *messageType* string describing the kind of message to post, should be one of...

- `kOfxMessageError`
- `kOfxMessageWarning`
- `kOfxMessageMessage`
- `messageId` plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- `format` printf style format string
- ... printf style varargs list to print

Persistent messages are associated with an effect handle until explicitly cleared by an effect. So if an error message is posted the error state, and associated message will persist and be displayed on the effect appropriately. (eg: draw a node in red on a node based compositor and display the message when clicked on).

If `messageType` is error or warning, associated error states should be flagged on host applications. Posting an error message implies that the host cannot proceed, a warning allows the host to proceed, whilst a simple message should have no stop anything.

Return

- `kOfxStatOK` - if the message was successfully posted
- `kOfxStatErrBadHandle` - the handle was rubbish
- `kOfxStatFailed` - if the message could not be posted for some reason

`OfxStatus (*clearPersistentMessage)(void *handle)`

Clears any persistent message on an effect handle that was set by `OfxMessageSuiteV2::setPersistentMessage`. New for V2 message suite.

- `handle` effect instance handle messages should be cleared from.
- `handle` effect handle (descriptor or instance)

Clearing a message will clear any associated error state.

Return

- `kOfxStatOK` - if the message was successfully cleared
- `kOfxStatErrBadHandle` - the handle was rubbish
- `kOfxStatFailed` - if the message could not be cleared for some reason

1.18.12 OfxImageEffectOpenGLRenderSuiteV1

struct `OfxImageEffectOpenGLRenderSuiteV1`

OFX suite that provides image to texture conversion for OpenGL processing.

Public Members

OfxStatus (***clipLoadTexture**)(*OfxImageClipHandle* clip, *OfxTime* time, const char *format, const *OfxRectD* *region, *OfxPropertySetHandle* *textureHandle)

loads an image from an OFX clip as a texture into OpenGL

- **clip** clip to load the image from
- **time** effect time to load the image from
- **format** requested texture format (As in none,byte,word,half,float, etc..) When set to NULL, the host decides the format based on the plug-in's *kOfxOpenGLPropPixelDepth* setting.
- **region** region of the image to load (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- **textureHandle** property set containing information about the texture

An image is fetched from a clip at the indicated time for the given region and loaded into an OpenGL texture. When a specific format is requested, the host ensures it gives the requested format. When the clip specified is the "Output" clip, the format is ignored and the host must bind the resulting texture as the current color buffer (render target). This may also be done prior to calling the *kOfxImageEffectActionRender* action. If the *region* parameter is set to non-NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set or is NULL, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped to the clip's Region of Definition. Information about the texture, including the texture index, is returned in the *textureHandle* argument. The properties on this handle will be...

- *kOfxImageEffectPropOpenGLTextureIndex*
- *kOfxImageEffectPropOpenGLTextureTarget*
- *kOfxImageEffectPropPixelDepth*
- *kOfxImageEffectPropComponents*
- *kOfxImageEffectPropPreMultiplication*
- *kOfxImageEffectPropRenderScale*
- *kOfxImagePropPixelAspectRatio*
- *kOfxImagePropBounds*
- *kOfxImagePropRegionOfDefinition*
- *kOfxImagePropRowBytes*
- *kOfxImagePropField*
- *kOfxImagePropUniqueIdentifier*

With the exception of the OpenGL specifics, these properties are the same as the properties in an image handle returned by *clipGetImage* in the image effect suite.

Note:

- this is the OpenGL equivalent of *clipGetImage* from *OfxImageEffectSuiteV1*
-

Pre

- clip was returned by clipGetHandle
- Format property in the texture handle

Post

- texture handle to be disposed of by clipFreeTexture before the action returns
- when the clip specified is the “Output” clip, the format is ignored and the host must bind the resulting texture as the current color buffer (render target). This may also be done prior to calling the render action.

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time and/or region, the plug-in should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - not enough OpenGL memory was available for the effect to load the texture. The plug-in should abort the GL render and return *kOfxStatErrMemory*, after which the host can decide to retry the operation with CPU based processing.

OfxStatus (*clipFreeTexture)(*OfxPropertySetHandle* textureHandle)

Releases the texture handle previously returned by clipLoadTexture.

For input clips, this also deletes the texture from OpenGL. This should also be called on the output clip; for the Output clip, it just releases the handle but does not delete the texture (since the host will need to read it).

Pre

- textureHandle was returned by clipGetImage

Post

- all operations on textureHandle will be invalid, and the OpenGL texture it referred to has been deleted (for source clips)

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - general failure for some reason,
- *kOfxStatErrBadHandle* - the image handle was invalid,

OfxStatus (*flushResources)()

Request the host to minimize its GPU resource load.

When a plug-in fails to allocate GPU resources, it can call this function to request the host to flush its GPU resources if it holds any. After the function the plug-in can try again to allocate resources which then might succeed if the host actually has released anything.

Pre**Post**

- No changes to the plug-in GL state should have been made.

Return

- *kOfxStatOK* - the host has actually released some resources,
- *kOfxStatReplyDefault* - nothing the host could do..

Properties for GPU rendering with other acceleration methods:

group **CudaRender**

Version

CUDA rendering was added in version 1.5.

Defines

kOfxImageEffectPropCudaRenderSupported

Indicates whether a host or plug-in can support CUDA render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support CUDA render
 - “true” - the host or plug-in can support CUDA render

kOfxImageEffectPropCudaEnabled

Indicates that a plug-in SHOULD use CUDA render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropCudaRenderSupported=“true”` then the host MAY set this property to indicate that it is passing images as CUDA memory pointers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the `kOfxImagePropData` of each image of each clip is a CPU memory pointer.
 - 1 indicates that the `kOfxImagePropData` of each image of each clip is a CUDA memory pointer.

kOfxImageEffectPropCudaStreamSupported

Indicates whether a host or plug-in can support CUDA streams.

- Type - string X 1

- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - in which case the host or plug-in does not support CUDA streams
 - “true” - which means a host or plug-in can support CUDA streams

kOfxImageEffectPropCudaStream

The CUDA stream to be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property will only be set if the host and plug-in both support CUDA streams.

If set:

- this property contains a pointer to the stream of CUDA render (`cudaStream_t`). In order to use it, `reinterpret_cast<cudaStream_t>(pointer)` is needed.
- the plug-in SHOULD ensure that its render action enqueues any asynchronous CUDA operations onto the supplied queue.
- the plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action, and SHOULD NOT call `cudaDeviceSynchronize()` at any time.

If not set:

- the plug-in SHOULD ensure that any asynchronous operations it enqueues have completed before returning from the render action.

group **MetalRender**

Version

Metal rendering was added in version 1.5.

Defines

kOfxImageEffectPropMetalRenderSupported

Indicates whether a host or plug-in can support Metal render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)

- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support Metal render
 - “true” - the host or plug-in can support Metal render

kOfxImageEffectPropMetalEnabled

Indicates that a plug-in SHOULD use Metal render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropMetalRenderSupported=“true”` then the host MAY set this property to indicate that it is passing images as Metal buffers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the `kOfxImagePropData` of each image of each clip is a CPU memory pointer.
 - 1 indicates that the `kOfxImagePropData` of each image of each clip is a Metal `id<MTLBuffer>`.

kOfxImageEffectPropMetalCommandQueue

The command queue of Metal render.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property contains a pointer to the command queue to be used for Metal rendering (`id<MTLCommandQueue>`). In order to use it, `reinterpret_cast<id<MTLCommandQueue>>(pointer)` is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous Metal operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

group **OpenCLRender**

Version

OpenCL rendering was added in version 1.5.

Defines

kOfxImageEffectPropOpenCLRenderSupported

Indicates whether a host or plug-in can support OpenCL Buffers render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support OpenCL Buffers render
 - “true” - the host or plug-in can support OpenCL Buffers render

kOfxImageEffectPropOpenCLSupported

Indicates whether a host or plug-in can support OpenCL Images render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - in which case the host or plug-in does not support OpenCL Images render
 - “true” - which means a host or plug-in can support OpenCL Images render

kOfxImageEffectPropOpenCLEnabled

Indicates that a plug-in SHOULD use OpenCL render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropOpenCLRenderSupported=“true”` or have both set `kOfxImageEffectPropOpenCLSupported=“true”` then the host MAY set this property to indicate that it is passing images as OpenCL Buffers or Images.

When rendering using OpenCL Buffers, the `cl_mem` of the buffers are retrieved using *kOfxImagePropData*. When rendering using OpenCL Images, the `cl_mem` of the images are retrieved using *kOfxImageEffectPropOpenCLImage*. If both *kOfxImageEffectPropOpenCLSupported* (Buffers) and *kOfxImageEffectPropOpenCLRenderSupported* (Images) are enabled by the plug-in, it should use *kOfxImageEffectPropOpenCLImage* to determine which is being used by the host.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that a plug-in SHOULD use OpenCL render in the render action

- 1 indicates that a plug-in SHOULD NOT use OpenCL render in the render action

kOfxImageEffectPropOpenCLCommandQueue

Indicates the OpenCL command queue that should be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property contains a pointer to the command queue to be used for OpenCL rendering (`cl_command_queue`). In order to use it, `reinterpret_cast<cl_command_queue>(pointer)` is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous OpenCL operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

kOfxImageEffectPropOpenCLImage

Indicates the image handle of an image supplied as an OpenCL Image by the host.

- Type - pointer X 1
- Property Set - image handle returned by `clipGetImage`

This value should be cast to a `cl_mem` and used as the image handle when performing OpenCL Images operations. The property should be used (not *kOfxImagePropData*) when rendering with OpenCL Images (*kOfxImageEffectPropOpenCLSupported*), and should be used to determine whether Images or Buffers should be used if a plug-in supports both *kOfxImageEffectPropOpenCLSupported* and *kOfxImageEffectPropOpenCLRenderSupported*. Note: the `kOfxImagePropRowBytes` property is not required to be set by the host, since OpenCL Images do not have the concept of row bytes.

kOfxOpenCLProgramSuite

Typedefs

typedef struct *OfxOpenCLProgramSuiteV1* **OfxOpenCLProgramSuiteV1**

OFX suite that allows a plug-in to get OpenCL programs compiled.

This is an optional suite the host can provide for building OpenCL programs for the plug-in, as an alternative to calling `clCreateProgramWithSource / clBuildProgram`. There are two advantages to doing this: The host can add flags (such as `-cl-denorms-are-zero`) to the build call, and may also cache program binaries for performance (however, if the source of the program or the OpenCL environment changes, the host must recompile so some mechanism such as hashing must be used).

struct **OfxOpenCLProgramSuiteV1**

#include <ofxGPURender.h> OFX suite that allows a plug-in to get OpenCL programs compiled.

This is an optional suite the host can provide for building OpenCL programs for the plug-in, as an alternative to calling `clCreateProgramWithSource / clBuildProgram`. There are two advantages to doing this: The host can add flags (such as `-cl-denorms-are-zero`) to the build call, and may also cache program binaries for performance (however, if the source of the program or the OpenCL environment changes, the host must recompile so some mechanism such as hashing must be used).

Public Members

OfxStatus (***compileProgram**)(const char *pszProgramSource, int fOptional, void *pResult)

Compiles the OpenCL program.

1.18.13 OfxDrawSuiteV1: Drawing Overlays

Added for OFX v1.5, Jan 2022.

See the source at [ofxDrawSuite.h](#)

struct **OfxDrawSuiteV1**

OFX suite that allows an effect to draw to a host-defined display context.

Public Members

OfxStatus (***getColor**)(*OfxDrawContextHandle* context, *OfxStandardColour* std_colour, *OfxRGBAColourF* *colour)

Retrieves the host's desired draw colour for.

- `context` draw context
- `std_colour` desired colour type
- `colour` returned RGBA colour

Return

- *kOfxStatOK* - the colour was returned
- *kOfxStatErrValue* - `std_colour` was invalid
- *kOfxStatFailed* - failure, e.g. if function is called outside `kOfxInteractActionDraw`

OfxStatus (***setColor**)(*OfxDrawContextHandle* context, const *OfxRGBAColourF* *colour)

Sets the colour for future drawing operations (lines, filled shapes and text)

- `context` draw context
- `colour` RGBA colour

The host should use “over” compositing when using a non-opaque colour.

Return

- *kOfxStatOK* - the colour was changed
- *kOfxStatFailed* - failure, e.g. if function is called outside *kOfxInteractActionDraw*

OfxStatus (***setLineWidth**)(*OfxDrawContextHandle* context, float width)

Sets the line width for future line drawing operations.

- context draw context
- width line width

Use width 0 for a single pixel line or non-zero for a smooth line of the desired width

The host should adjust for screen density.

Return

- *kOfxStatOK* - the width was changed
- *kOfxStatFailed* - failure, e.g. if function is called outside *kOfxInteractActionDraw*

OfxStatus (***setLineStipple**)(*OfxDrawContextHandle* context, *OfxDrawLineStipplePattern* pattern)

Sets the stipple pattern for future line drawing operations.

- context draw context
- pattern desired stipple pattern

Return

- *kOfxStatOK* - the pattern was changed
- *kOfxStatErrValue* - pattern was not valid
- *kOfxStatFailed* - failure, e.g. if function is called outside *kOfxInteractActionDraw*

OfxStatus (***draw**)(*OfxDrawContextHandle* context, *OfxDrawPrimitive* primitive, const *OfxPointD* *points, int point_count)

Draws a primitive of the desired type.

- context draw context
- primitive desired primitive
- points array of points in the primitive
- point_count number of points in the array

kOfxDrawPrimitiveLines - like *GL_LINES*, n points draws n/2 separated lines
kOfxDrawPrimitiveLineStrip - like *GL_LINE_STRIP*, n points draws n-1 connected lines
kOfxDrawPrimitiveLineLoop - like *GL_LINE_LOOP*, n points draws n connected lines
kOfxDrawPrimitiveRectangle - draws an axis-aligned filled rectangle defined by 2 opposite corner points
kOfxDrawPrimitivePolygon - like *GL_POLYGON*, draws a filled n-sided polygon
kOfxDrawPrimitiveEllipse - draws a axis-aligned elliptical line (not filled) within the rectangle defined by 2 opposite corner points

Return

- *kOfxStatOK* - the draw was completed
- *kOfxStatErrValue* - invalid primitive, or point_count not valid for primitive
- *kOfxStatFailed* - failure, e.g. if function is called outside kOfxInteractActionDraw

OfxStatus (***drawText**)(*OfxDrawContextHandle* context, const char *text, const *OfxPointD* *pos, int alignment)

Draws text at the specified position.

- **context** draw context
- **text** text to draw (UTF-8 encoded)
- **pos** position at which to align the text
- **alignment** text alignment flags (see kOfxDrawTextAlignment*)

The text font face and size are determined by the host.

Return

- *kOfxStatOK* - the text was drawn
- *kOfxStatErrValue* - text or pos were not defined
- *kOfxStatFailed* - failure, e.g. if function is called outside kOfxInteractActionDraw

#defines**kOfxInteractPropDrawContext**

The Draw Context handle.

- Type - pointer X 1
- Property Set - read only property on the inArgs of the following actions...
- *kOfxInteractActionDraw*

Enumsenum **OfxStandardColour**

Defines valid values for *OfxDrawSuiteV1::getColour*.

Values:

enumerator **kOfxStandardColourOverlayBackground**

enumerator **kOfxStandardColourOverlayActive**

enumerator **kOfxStandardColourOverlaySelected**

enumerator **kOfxStandardColourOverlayDeselected**

enumerator **kOfxStandardColourOverlayMarqueeFG**

enumerator **kOfxStandardColourOverlayMarqueeBG**

enumerator **kOfxStandardColourOverlayText**

enum **OfxDrawLineStipplePattern**

Defines valid values for *OfxDrawSuiteV1::setLineStipple*.

Values:

enumerator **kOfxDrawLineStipplePatternSolid**

enumerator **kOfxDrawLineStipplePatternDot**

enumerator **kOfxDrawLineStipplePatternDash**

enumerator **kOfxDrawLineStipplePatternAltDash**

enumerator **kOfxDrawLineStipplePatternDotDash**

enum **OfxDrawPrimitive**

Defines valid values for *OfxDrawSuiteV1::draw*.

Values:

enumerator **kOfxDrawPrimitiveLines**

enumerator **kOfxDrawPrimitiveLineStrip**

enumerator **kOfxDrawPrimitiveLineLoop**

enumerator **kOfxDrawPrimitiveRectangle**

enumerator **kOfxDrawPrimitivePolygon**

enumerator **kOfxDrawPrimitiveEllipse**

<p>Warning: doxygenenum: Cannot find enum “OfxDrawTextAligment” in doxygen xml output for project “ofx_reference” from directory: ../doxygen_build/xml/</p>
--

Warning: This section is outdated and should be properly generated automatically from source code instead of maintaining it aside

1.19 Properties by object reference

1.19.1 Properties on the Image Effect Host

- `kOfxPropAPIVersion` - (read only) the version of the API implemented by the host. If not present, it is safe to assume “1.0”
- `kOfxPropType` - (read only) set to “host”
- `kOfxPropName` - (read only) the globally unique name of the application, eg: “com.acmesoftware.funkyCompositor”
- `kOfxPropLabel` - (read only) the user visible name of the application,
- `kOfxPropVersion` - (read only) the version number of the host
- `kOfxPropVersionLabel` - (read only) a user readable version label
- `kOfxImageEffectHostPropIsBackground` - (read only) is the application a background rendererr
- `kOfxImageEffectPropSupportsOverlays` - (read only) does the application support overlay interactive GUIs
- `kOfxImageEffectPropSupportsMultiResolution` - (read only) does the application support images of different sizes
- `kOfxImageEffectPropSupportsTiles` - (read only) does the application support image tiling
- `kOfxImageEffectPropTemporalClipAccess` - (read only) does the application allow random temporal access to source images
- `kOfxImageEffectPropSupportedComponents` - (read only) a list of supported colour components
- `kOfxImageEffectPropSupportedContexts` - (read only) a list of supported effect contexts
- `kOfxImageEffectPropSupportsMultipleClipDepths` - (read only) does the application allow inputs and output clips to have differing bit depths
- `kOfxImageEffectPropSupportsMultipleClipPARs` - (read only) does the application allow inputs and output clips to have differing pixel aspect ratios
- `kOfxImageEffectPropSetableFrameRate` - (read only) does the application allow an effect to change the frame rate of the output clip
- `kOfxImageEffectPropSetableFielding` - (read only) does the application allow an effect to change the fielding of the output clip
- `kOfxParamHostPropSupportsCustomInteract` - (read only) does the application
- `kOfxParamHostPropSupportsStringAnimation` - (read only) does the application allow the animation of string parameters
- `kOfxParamHostPropSupportsChoiceAnimation` - (read only) does the application allow the animation of choice parameters
- `kOfxParamHostPropSupportsBooleanAnimation` - (read only does the application allow the animation of boolean parameters)

- `kOfxParamHostPropSupportsCustomAnimation` - (read only) does the application allow the animation of custom parameters
- `kOfxParamHostPropMaxParameters` - (read only) the maximum number of parameters the application allows a plug-in to have
- `kOfxParamHostPropMaxPages` - (read only) the maximum number of parameter pages the application allows a plug-in to have
- `kOfxParamHostPropPageRowColumnCount` - (read only) the number of rows and columns on a page parameter
- `kOfxPropHostOSHandle` - (read only) a pointer to an OS specific application handle (eg: the root `hWnd` on Windows)
- `kOfxParamHostPropSupportsParametricAnimation` - (read only) does the host support animation of parametric parameters
- `kOfxImageEffectInstancePropSequentialRender` - (read only) does the host support sequential rendering
- `kOfxImageEffectPropOpenGLRenderSupported` - (read only) does the host support OpenGL accelerated rendering
- `kOfxImageEffectPropRenderQualityDraft` - (read only) does the host support draft quality rendering
- `kOfxImageEffectHostPropNativeOrigin` - (read only) native origin of the host

1.19.2 Properties on an Effect Descriptor

An image effect plugin (ie: that thing passed to the initial ‘describe’ action) has the following properties, these can only be set inside the ‘describe’ actions ...

- `kOfxPropType` - (read only)
- `kOfxPropLabel` - (read/write)
- `kOfxPropShortLabel` - (read/write)
- `kOfxPropLongLabel` - (read/write)
- `kOfxPropVersion` - (read only) the version number of the plugin
- `kOfxPropVersionLabel` - (read only) a user readable version label
- `kOfxPropPluginDescription` - (read/write), a short description of the plugin
- `kOfxImageEffectPropSupportedContexts` - (read/write)
- `kOfxImageEffectPluginPropGrouping` - (read/write)
- `kOfxImageEffectPluginPropSingleInstance` - (read/write)
- `kOfxImageEffectPluginRenderThreadSafety` - (read/write)
- `kOfxImageEffectPluginPropHostFrameThreading` - (read/write)
- `kOfxImageEffectPluginPropOverlayInteractV1` - (read/write)
- `kOfxImageEffectPropSupportsMultiResolution` - (read/write)
- `kOfxImageEffectPropSupportsTiles` - (read/write)
- `kOfxImageEffectPropTemporalClipAccess` - (read/write)
- `kOfxImageEffectPropSupportedPixelDepths` - (read/write)
- `kOfxImageEffectPluginPropFieldRenderTwiceAlways` - (read/write)

- kOfxImageEffectPropSupportsMultipleClipDepths - (read/write)
- kOfxImageEffectPropSupportsMultipleClipPARs - (read/write)
- kOfxImageEffectPluginRenderThreadSafety - (read/write)
- kOfxImageEffectPropClipPreferencesSlaveParam - (read/write)
- kOfxImageEffectPropOpenGLRenderSupported - (read and write)
- kOfxPluginPropFilePath (read only)

1.19.3 Properties on an Effect Instance

An image effect instance has the following properties, all but kOfxPropInstanceData and kOfxImageEffectInstancePropSequentialRender are read only...

- kOfxPropType - (read only)
- kOfxImageEffectPropContext - (read only)
- kOfxPropInstanceData - (read and write)
- kOfxImageEffectPropProjectSize - (read only)
- kOfxImageEffectPropProjectOffset - (read only)
- kOfxImageEffectPropProjectExtent - (read only)
- kOfxImageEffectPropProjectPixelAspectRatio - (read only)
- kOfxImageEffectInstancePropEffectDuration - (read only)
- kOfxImageEffectInstancePropSequentialRender - (read and write)
- kOfxImageEffectPropSupportsTiles - (read/write)
- kOfxImageEffectPropOpenGLRenderSupported - (read and write)
- kOfxImageEffectPropFrameRate - (read only)
- kOfxPropIsInteractive - (read only)

1.19.4 Properties on a Clip Descriptor

All OfxImageClipHandle accessed inside the kOfxActionDescribe or kOfxActionDescribeInContext are clip descriptors, used to describe the behaviour of clips in a specific context.

- kOfxPropType - (read only) set to
- kOfxPropName - (read only) the name the clip was created with
- kOfxPropLabel - (read/write) the user visible label for the clip
- kOfxPropShortLabel - (read/write)
- kOfxPropLongLabel - (read/write)
- kOfxImageEffectPropSupportedComponents - (read/write)
- kOfxImageEffectPropTemporalClipAccess - (read/write)
- kOfxImageClipPropOptional - (read/write)
- kOfxImageClipPropFieldExtraction - (read/write)

- kOfxImageClipPropIsMask - (read/write)
- kOfxImageEffectPropSupportsTiles - (read/write)

1.19.5 Properties on a Clip Instance

- kOfxPropType - (read only)
- kOfxPropName - (read only)
- kOfxPropLabel - (read only)
- kOfxPropShortLabel - (read only)
- kOfxPropLongLabel - (read only)
- kOfxImageEffectPropSupportedComponents - (read only)
- kOfxImageEffectPropTemporalClipAccess - (read only)
- kOfxImageClipPropOptional - (read only)
- kOfxImageClipPropFieldExtraction - (read only)
- kOfxImageClipPropIsMask - (read only)
- kOfxImageEffectPropSupportsTiles - (read only)
- kOfxImageEffectPropPixelDepth - (read only)
- kOfxImageEffectPropComponents - (read only)
- kOfxImageClipPropUnmappedPixelDepth - (read only)
- kOfxImageClipPropUnmappedComponents - (read only)
- kOfxImageEffectPropPreMultiplication - (read only)
- kOfxImagePropPixelAspectRatio - (read only)
- kOfxImageEffectPropFrameRate - (read only)
- kOfxImageEffectPropFrameRange - (read only)
- kOfxImageClipPropFieldOrder - (read only)
- kOfxImageClipPropConnected - (read only)
- kOfxImageEffectPropUnmappedFrameRange - (read only)*
- kOfxImageEffectPropUnmappedFrameRate - (read only)*
- kOfxImageClipPropContinuousSamples - (read only)

1.19.6 Properties on an Image

All images are instances, there is no such thing as an image descriptor.

- kOfxPropType - (read only)
- kOfxImageEffectPropPixelDepth - (read only)
- kOfxImageEffectPropComponents - (read only)
- kOfxImageEffectPropPreMultiplication - (read only)
- kOfxImageEffectPropRenderScale - (read only)

- kOfxImagePropPixelAspectRatio - (read only)
- kOfxImagePropData - (read only)
- kOfxImagePropBounds - (read only)
- kOfxImagePropRegionOfDefinition - (read only) *
- kOfxImagePropRowBytes - (read only)
- kOfxImagePropField - (read only)
- kOfxImagePropUniqueIdentifier - (read only)

1.19.7 Properties on Parameter Set Instances

kOfxPropParamSetNeedsSyncing , which indicates if private data is dirty and may need re-syncing to a parameter set
 .. ParameterProperties:

1.19.8 Properties on Parameter Descriptors and Instances

1.19.9 Properties Common to All Parameters

The following properties are common to all parameters...

- kOfxPropType , which will always be kOfxTypeParameter (read only)
- kOfxPropName read/write in the descriptor, read only on an instance
- kOfxPropLabel read/write in the descriptor and instance
- kOfxPropShortLabel read/write in the descriptor and instance
- kOfxPropLongLabel read/write in the descriptor and instance
- kOfxParamPropType read only in the descriptor and instance, the value is set on construction
- kOfxParamPropSecret read/write in the descriptor and instance
- kOfxParamPropHint read/write in the descriptor and instance
- kOfxParamPropScriptName read/write in the descriptor, read only on an instance
- kOfxParamPropParent read/write in the descriptor, read only on an instance
- kOfxParamPropEnabled read/write in the descriptor and instance
- kOfxParamPropDataPtr read/write in the descriptor and instance
- kOfxPropIcon , read/write on a descriptor, read only on an instance

1.19.10 Properties On Group Parameters

- kOfxParamPropGroupOpen read/write in the descriptor, read only on an instance

1.19.11 Properties Common to All But Group and Page Parameters

- kOfxParamPropInteractV1 read/write in the descriptor, read only on an instance
- kOfxParamPropInteractSize read/write in the descriptor, read only on an instance
- kOfxParamPropInteractSizeAspect read/write in the descriptor, read only on an instance
- kOfxParamPropInteractMinimumSize read/write in the descriptor, read only on an instance
- kOfxParamPropInteractPreferedSize read/write in the descriptor, read only on an instance
- kOfxParamPropHasHostOverlayHandle read only in the descriptor and instance
- kOfxParamPropUseHostOverlayHandle read/write in the descriptor and read only in the instance

1.19.12 Properties Common to All Parameters That Hold Values

- kOfxParamPropDefault read/write in the descriptor, read only on an instance
- kOfxParamPropAnimates read/write in the descriptor, read only on an instance
- kOfxParamPropIsAnimating read/write in the descriptor, read only on an instance
- kOfxParamPropIsAutoKeying read/write in the descriptor, read only on an instance
- kOfxParamPropPersistant read/write in the descriptor, read only on an instance
- kOfxParamPropEvaluateOnChange read/write in the descriptor and instance
- kOfxParamPropPluginMayWrite read/write in the descriptor, read only on an instance
- kOfxParamPropCacheInvalidation read/write in the descriptor, read only on an instance
- kOfxParamPropCanUndo read/write in the descriptor, read only on an instance

1.19.13 Properties Common to All Numeric Parameters

- kOfxParamPropMin read/write in the descriptor and instance
- kOfxParamPropMax read/write in the descriptor and instance
- kOfxParamPropDisplayMin read/write in the descriptor and instance
- kOfxParamPropDisplayMax read/write in the descriptor and instance

1.19.14 Properties Common to All Double Parameters

- kOfxParamPropIncrement read/write in the descriptor and instance
- kOfxParamPropDigits read/write in the descriptor and instance

1.19.15 Properties On 1D Double Parameters

- kOfxParamPropShowTimeMarker read/write in the descriptor and instance
- kOfxParamPropDoubleType read/write in the descriptor, read only on an instance

1.19.16 Properties On 2D and 3D Double Parameters

- kOfxParamPropDoubleType read/write in the descriptor, read only on an instance

1.19.17 Properties On Non Normalised Spatial Double Parameters

- kOfxParamPropDefaultCoordinateSystem read/write in the descriptor, read only on an instance

1.19.18 Properties On 2D and 3D Integer Parameters

- kOfxParamPropDimensionLabel read/write in the descriptor, read only on an instance

1.19.19 Properties On String Parameters

- kOfxParamPropStringMode read/write in the descriptor, read only on an instance
- kOfxParamPropStringFilePathExists read/write in the descriptor, read only on an instance

1.19.20 Properties On Choice Parameters

- kOfxParamPropChoiceOption read/write in the descriptor and instance
- kOfxParamPropChoiceOrder read/write in the descriptor and instance

1.19.21 Properties On Custom Parameters

- kOfxParamPropCustomInterpCallbackV1 read/write in the descriptor, read only on an instance

1.19.22 Properties On Page Parameters

- kOfxParamPropPageChild read/write in the descriptor, read only on an instance

1.19.23 On Parametric Parameters

- kOfxParamPropAnimates read/write in the descriptor, read only on an instance
- kOfxParamPropIsAnimating read/write in the descriptor, read only on an instance
- kOfxParamPropIsAutoKeying read/write in the descriptor, read only on an instance
- kOfxParamPropPersistant read/write in the descriptor, read only on an instance
- kOfxParamPropEvaluateOnChange read/write in the descriptor and instance
- kOfxParamPropPluginMayWrite read/write in the descriptor, read only on an instance
- kOfxParamPropCacheInvalidation read/write in the descriptor, read only on an instance
- kOfxParamPropCanUndo read/write in the descriptor, read only on an instance
- kOfxParamPropParametricDimension read/write in the descriptor, read only on an instance
- kOfxParamPropParametricUIColour read/write in the descriptor, read only on an instance
- kOfxParamPropParametricInteractBackground read/write in the descriptor, read only on an instance
- kOfxParamPropParametricRange read/write in the descriptor, read only on an instance

1.19.24 Properties on Interact Descriptors

- kOfxInteractPropHasAlpha read only
- kOfxInteractPropBitDepth read only

1.19.25 Properties on Interact Instances

- kOfxPropEffectInstance read only
- kOfxPropInstanceData read/write only
- kOfxInteractPropPixelScale read only
- kOfxInteractPropBackgroundColour read only
- kOfxInteractPropHasAlpha read only
- kOfxInteractPropBitDepth read only
- kOfxInteractPropSlaveToParam read/write
- kOfxInteractPropSuggestedColour read only

1.20 Properties Reference

kOfxImageClipPropConnected

Says whether the clip is actually connected at the moment.

- Type - int X 1
- Property Set - clip instance (read only)
- Valid Values - This must be one of 0 or 1

An instance may have a clip may not be connected to an object that can produce image data. Use this to find out.

Any clip that is not optional will *always* be connected during a render action. However, during interface actions, even non optional clips may be unconnected.

kOfxImageClipPropContinuousSamples

Clip and action argument property which indicates that the clip can be sampled continuously.

- Type - int X 1
- Property Set - clip instance (read only), as an out argument to *kOfxImageEffectActionGetClipPreferences* action (read/write)
- Default - 0 as an out argument to the *kOfxImageEffectActionGetClipPreferences* action
- Valid Values - This must be one of...
 - 0 if the images can only be sampled at discreet times (eg: the clip is a sequence of frames),
 - 1 if the images can only be sampled continuously (eg: the clip is infact an animating roto spline and can be rendered anywhen).

If this is set to true, then the frame rate of a clip is effectively infinite, so to stop arithmetic errors the frame rate should then be set to 0.

kOfxImageClipPropFieldExtraction

Controls how a plugin fetched fielded imagery from a clip.

- Type - string X 1
- Property Set - a clip descriptor (read/write)
- Default - kOfxImageFieldDoubled
- Valid Values - This must be one of
 - kOfxImageFieldBoth - fetch a full frame interlaced image
 - kOfxImageFieldSingle - fetch a single field, making a half height image
 - kOfxImageFieldDoubled - fetch a single field, but doubling each line and so making a full height image

This controls how a plug-in wishes to fetch images from a fielded clip, so it can tune it behaviour when it renders fielded footage.

Note that if it fetches `kOfxImageFieldSingle` and the host stores images natively as both fields interlaced, it can return a single image by doubling rowbytes and tweaking the starting address of the image data. This saves on a buffer copy.

kOfxImageClipPropFieldOrder

Which spatial field occurs temporally first in a frame.

- Type - string X 1
- Property Set - a clip instance (read only)
- Valid Values - This must be one of
 - *kOfxImageFieldNone* - the material is unfielded
 - *kOfxImageFieldLower* - the material is fielded, with image rows 0,2,4... occurring first in a frame
 - *kOfxImageFieldUpper* - the material is fielded, with image rows line 1,3,5... occurring first in a frame

kOfxImageClipPropIsMask

Indicates that a clip is intended to be used as a mask input.

- Type - int X 1
- Property Set - clip descriptor (read/write)
- Default - 0
- Valid Values - This must be one of 0 or 1

Set this property on any clip which will only ever have single channel alpha images fetched from it. Typically on an optional clip such as a junk matte in a keyer.

This property acts as a hint to hosts indicating that they could feed the effect from a rotoshape (or similar) rather than an 'ordinary' clip.

kOfxImageClipPropOptional

Indicates if a clip is optional.

- Type - int X 1
- Property Set - clip descriptor (read/write)
- Default - 0
- Valid Values - This must be one of 0 or 1

kOfxImageClipPropUnmappedComponents

Indicates the current 'raw' component type on a clip before any mapping by clip preferences.

- Type - string X 1
- Property Set - clip instance (read only),
- Valid Values - This must be one of

- kOfxImageComponentNone (implying a clip is unconnected)
- kOfxImageComponentRGBA
- kOfxImageComponentRGB
- kOfxImageComponentAlpha

kOfxImageClipPropUnmappedPixelDepth

Indicates the type of each component in a clip before any mapping by clip preferences.

- Type - string X 1
- Property Set - clip instance (read only)
- Valid Values - This must be one of
 - kOfxBitDepthNone (implying a clip is unconnected image)
 - kOfxBitDepthByte
 - kOfxBitDepthShort
 - kOfxBitDepthHalf
 - kOfxBitDepthFloat

This is the actual value of the component depth, before any mapping by clip preferences.

kOfxImageEffectFrameVarying

Indicates whether an effect will generate different images from frame to frame.

- Type - int X 1
- Property Set - out argument to *kOfxImageEffectActionGetClipPreferences* action (read/write).
- Default - 0
- Valid Values - This must be one of 0 or 1

This property indicates whether a plugin will generate a different image from frame to frame, even if no parameters or input image changes. For example a generator that creates random noise pixel at each frame.

kOfxImageEffectHostPropIsBackground

Indicates if a host is a background render.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 if the host is a foreground host, it may open the effect in an interactive session (or not)
 - 1 if the host is a background 'processing only' host, and the effect will never be opened in an interactive session.

kOfxImageEffectInstancePropEffectDuration

The duration of the effect.

- Type - double X 1
- Property Set - a plugin instance (read only)

This contains the duration of the plug-in effect, in frames.

kOfxImageEffectInstancePropSequentialRender

Indicates whether a plugin needs sequential rendering, and a host support it.

- Type - int X 1
- Property Set - plugin descriptor (read/write) or plugin instance (read/write), and host descriptor (read only)
- Default - 0
- Valid Values -
 - 0 - for a plugin, indicates that a plugin does not need to be sequentially rendered to be correct, for a host, indicates that it cannot ever guarantee sequential rendering,
 - 1 - for a plugin, indicates that it needs to be sequentially rendered to be correct, for a host, indicates that it can always support sequential rendering of plugins that are sequentially rendered,
 - 2 - for a plugin, indicates that it is best to render sequentially, but will still produce correct results if not, for a host, indicates that it can sometimes render sequentially, and will have set *kOfxImageEffectPropSequentialRenderStatus* on the relevant actions

Some effects have temporal dependencies, some information from the rendering of frame N-1 is needed to render frame N correctly. This property is set by an effect to indicate such a situation. Also, some effects are more efficient if they run sequentially, but can still render correct images even if they do not, eg: a complex particle system.

During an interactive session a host may attempt to render a frame out of sequence (for example when the user scrubs the current time), and the effect needs to deal with such a situation as best it can to provide feedback to the user.

However if a host caches output, any frame generated in random temporal order needs to be considered invalid and needs to be re-rendered when the host finally performs a first to last render of the output sequence.

In all cases, a host will set the *kOfxImageEffectPropSequentialRenderStatus* flag to indicate its sequential render status.

kOfxImageEffectPluginPropFieldRenderTwiceAlways

Controls how a plugin renders fielded footage.

- Type - integer X 1
- Property Set - a plugin descriptor (read/write)
- Default - 1
- Valid Values - This must be one of

- 0 - the plugin is to have its render function called twice, only if there is animation in any of its parameters
- 1 - the plugin is to have its render function called twice always

kOfxImageEffectPluginPropGrouping

Indicates the effect group for this plugin.

- Type - UTF8 string X 1
- Property Set - plugin descriptor (read/write)
- Default - ""

This is purely a user interface hint for the host so it can group related effects on any menus it may have.

kOfxImageEffectPluginPropHostFrameThreading

Indicates whether a plugin lets the host perform per frame SMP threading.

- Type - int X 1
- Property Set - plugin descriptor (read/write)
- Default - 1
- Valid Values - This must be one of
 - 0 - which means that the plugin will perform any per frame SMP threading
 - 1 - which means the host can call an instance's render function simultaneously at the same frame, but with different windows to render.

kOfxImageEffectPluginPropOverlayInteractV1

Sets the entry for an effect's overlay interaction.

- Type - pointer X 1
- Property Set - plugin descriptor (read/write)
- Default - NULL
- Valid Values - must point to an *OfxPluginEntryPoint*

The entry point pointed to must be one that handles custom interaction actions.

kOfxImageEffectPluginPropOverlayInteractV2

Sets the entry for an effect's overlay interaction. Unlike **kOfxImageEffectPluginPropOverlayInteractV1**, the overlay interact in the plug-in is expected to implement the **kOfxInteractActionDraw** using the *OfxDrawSuiteV1*.

- Type - pointer X 1
- Property Set - plugin descriptor (read/write)
- Default - NULL
- Valid Values - must point to an *OfxPluginEntryPoint*

The entry point pointed to must be one that handles custom interaction actions.

kOfxImageEffectPluginPropSingleInstance

Indicates whether only one instance of a plugin can exist at the same time.

- Type - int X 1
- Property Set - plugin descriptor (read/write)
- Default - 0
- Valid Values - This must be one of
 - 0 - which means multiple instances can exist simultaneously,
 - 1 - which means only one instance can exist at any one time.

Some plugins, for whatever reason, may only be able to have a single instance in existence at any one time. This plugin property is used to indicate that.

kOfxImageEffectPluginRenderThreadSafety

Indicates how many simultaneous renders the plugin can deal with.

- Type - string X 1
- Property Set - plugin descriptor (read/write)
- Default - *kOfxImageEffectRenderInstanceSafe*
- Valid Values - This must be one of
 - *kOfxImageEffectRenderUnsafe* - indicating that only a single ‘render’ call can be made at any time among all instances,
 - *kOfxImageEffectRenderInstanceSafe* - indicating that any instance can have a single ‘render’ call at any one time,
 - *kOfxImageEffectRenderFullySafe* - indicating that any instance of a plugin can have multiple renders running simultaneously

kOfxImageEffectPropClipPreferencesSlaveParam

Indicates the set of parameters on which a value change will trigger a change to clip preferences.

- Type - string X N
- Property Set - plugin descriptor (read/write)
- Default - none set
- Valid Values - the name of any described parameter

The plugin uses this to inform the host of the subset of parameters that affect the effect’s clip preferences. A value change in any one of these will trigger a call to the clip preferences action.

The plugin can be slaved to multiple parameters (setting index 0, then index 1 etc...)

kOfxImageEffectPropComponents

Indicates the current component type in a clip or image (after any mapping)

- Type - string X 1
- Property Set - clip instance (read only), image instance (read only)
- Valid Values - This must be one of
 - kOfxImageComponentNone (implying a clip is unconnected, not valid for an image)
 - kOfxImageComponentRGBA
 - kOfxImageComponentRGB
 - kOfxImageComponentAlpha

Note that for a clip, this is the value set by the clip preferences action, not the raw ‘actual’ value of the clip.

kOfxImageEffectPropContext

Indicates the context a plugin instance has been created for.

- Type - string X 1
- Property Set - image effect instance (read only)
- Valid Values - This must be one of
 - *kOfxImageEffectContextGenerator*
 - *kOfxImageEffectContextFilter*
 - *kOfxImageEffectContextTransition*
 - *kOfxImageEffectContextPaint*
 - *kOfxImageEffectContextGeneral*
 - *kOfxImageEffectContextRetimer*

kOfxImageEffectPropCudaEnabled

Indicates that a plug-in SHOULD use CUDA render in the current action.

If a plug-in and host have both set kOfxImageEffectPropCudaRenderSupported=”true” then the host MAY set this property to indicate that it is passing images as CUDA memory pointers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the kOfxImagePropData of each image of each clip is a CPU memory pointer.

- 1 indicates that the `kOfxImagePropData` of each image of each clip is a CUDA memory pointer.

`kOfxImageEffectPropCudaRenderSupported`

Indicates whether a host or plug-in can support CUDA render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support CUDA render
 - “true” - the host or plug-in can support CUDA render

`kOfxImageEffectPropCudaStream`

The CUDA stream to be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *`kOfxImageEffectActionRender`*
 - *`kOfxImageEffectActionBeginSequenceRender`*
 - *`kOfxImageEffectActionEndSequenceRender`*

This property will only be set if the host and plug-in both support CUDA streams.

If set:

- this property contains a pointer to the stream of CUDA render (`cudaStream_t`). In order to use it, `reinterpret_cast<cudaStream_t>(pointer)` is needed.
- the plug-in SHOULD ensure that its render action enqueues any asynchronous CUDA operations onto the supplied queue.
- the plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action, and SHOULD NOT call `cudaDeviceSynchronize()` at any time.

If not set:

- the plug-in SHOULD ensure that any asynchronous operations it enqueues have completed before returning from the render action.

`kOfxImageEffectPropCudaStreamSupported`

Indicates whether a host or plug-in can support CUDA streams.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)

- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - in which case the host or plug-in does not support CUDA streams
 - “true” - which means a host or plug-in can support CUDA streams

kOfxImageEffectPropFieldToRender

Indicates which field is being rendered.

- Type - string X 1
- Property Set - a read only in argument property to *kOfxImageEffectActionRender* and *kOfxImageEffectActionIsIdentity*
- Valid Values - this must be one of
 - kOfxImageFieldNone - there are no fields to deal with, all images are full frame
 - kOfxImageFieldBoth - the imagery is fielded and both scan lines should be rendered
 - kOfxImageFieldLower - the lower field is being rendered (lines 0,2,4...)
 - kOfxImageFieldUpper - the upper field is being rendered (lines 1,3,5...)

kOfxImageEffectPropFrameRange

The frame range over which a clip has images.

- Type - double X 2
- Property Set - clip instance (read only)

Dimension 0 is the first frame for which the clip can produce valid data.

Dimension 1 is the last frame for which the clip can produce valid data.

kOfxImageEffectPropFrameRate

The frame rate of a clip or instance’s project.

- Type - double X 1
- Property Set - clip instance (read only), effect instance (read only) and *kOfxImageEffectActionGetClipPreferences* action out args property (read/write)

For an input clip this is the frame rate of the clip.

For an output clip, the frame rate mapped via pixel preferences.

For an instance, this is the frame rate of the project the effect is in.

For the outargs property in the *kOfxImageEffectActionGetClipPreferences* action, it is used to change the frame rate of the output clip.

kOfxImageEffectPropFrameStep

The frame step used for a sequence of renders.

- Type - double X 1
- Property Set - an in argument for the *kOfxImageEffectActionBeginSequenceRender* action (read only)
- Valid Values - can be any positive value, but typically
 - 1 for frame based material
 - 0.5 for field based material

kOfxImageEffectPropInAnalysis

Indicates whether an effect is performing an analysis pass. `—ofxImageEffects.h`.

- Type - int X 1
- Property Set - plugin instance (read/write)
- Default - to 0
- Valid Values - This must be one of 0 or 1

Deprecated:

- This feature has been deprecated - officially commented out v1.4.

kOfxImageEffectPropInteractiveRenderStatus

Property that indicates if a plugin is being rendered in response to user interaction.

- Type - int X 1
- Property Set - read only property on the inArgs of the following actions...
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values -
 - 0 - the host is rendering the instance due to some reason other than an interactive tweak on a UI,
 - 1 - the instance is being rendered because a user is modifying parameters in an interactive session.

This property is set to 1 on all render calls that have been triggered because a user is actively modifying an effect (or up stream effect) in an interactive session. This typically means that the effect is not being rendered as a part of a sequence, but as a single frame.

kOfxImageEffectPropMetalCommandQueue

The command queue of Metal render.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*

– *kOfxImageEffectActionEndSequenceRender*

This property contains a pointer to the command queue to be used for Metal rendering (id<MTLCommandQueue>). In order to use it, reinterpret_cast<id<MTLCommandQueue>>(pointer) is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous Metal operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

kOfxImageEffectPropMetalEnabled

Indicates that a plug-in SHOULD use Metal render in the current action.

If a plug-in and host have both set kOfxImageEffectPropMetalRenderSupported="true" then the host MAY set this property to indicate that it is passing images as Metal buffers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the kOfxImagePropData of each image of each clip is a CPU memory pointer.
 - 1 indicates that the kOfxImagePropData of each image of each clip is a Metal id<MTLBuffer>.

kOfxImageEffectPropMetalRenderSupported

Indicates whether a host or plug-in can support Metal render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - "false" for a plug-in
- Valid Values - This must be one of
 - "false" - the host or plug-in does not support Metal render
 - "true" - the host or plug-in can support Metal render

kOfxImageEffectPropOpenCLCommandQueue

Indicates the OpenCL command queue that should be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*

- *kOfxImageEffectActionEndSequenceRender*

This property contains a pointer to the command queue to be used for OpenCL rendering (`cl_command_queue`). In order to use it, `reinterpret_cast<cl_command_queue>(pointer)` is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous OpenCL operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

kOfxImageEffectPropOpenCLEnabled

Indicates that a plug-in SHOULD use OpenCL render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropOpenCLRenderSupported="true"` or have both set `kOfxImageEffectPropOpenCLSupported="true"` then the host MAY set this property to indicate that it is passing images as OpenCL Buffers or Images.

When rendering using OpenCL Buffers, the `cl_mem` of the buffers are retrieved using *kOfxImagePropData*. When rendering using OpenCL Images, the `cl_mem` of the images are retrieved using *kOfxImageEffectPropOpenCLImage*. If both *kOfxImageEffectPropOpenCLSupported* (Buffers) and *kOfxImageEffectPropOpenCLRenderSupported* (Images) are enabled by the plug-in, it should use *kOfxImageEffectPropOpenCLImage* to determine which is being used by the host.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that a plug-in SHOULD use OpenCL render in the render action
 - 1 indicates that a plug-in SHOULD NOT use OpenCL render in the render action

kOfxImageEffectPropOpenCLRenderSupported

Indicates whether a host or plug-in can support OpenCL Buffers render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support OpenCL Buffers render
 - “true” - the host or plug-in can support OpenCL Buffers render

kOfxImageEffectPropOpenGLEnabled

Indicates that a plug-in SHOULD use OpenGL acceleration in the current action.

When a plug-in and host have established they can both use OpenGL renders then when this property has been set the host expects the plug-in to render its result into the buffer it has setup before calling the render. The plug-in can then also safely use the 'OfxImageEffectOpenGLRenderSuite'

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the plug-in cannot use the OpenGL suite
 - 1 indicates that the plug-in should render into the texture, and may use the OpenGL suite functions.

v1.4: kOfxImageEffectPropOpenGLEnabled should probably be checked in Instance Changed prior to try to read image via clipLoadTexture

Note: Once this property is set, the host and plug-in have agreed to use OpenGL, so the effect SHOULD access all its images through the OpenGL suite.

kOfxImageEffectPropOpenGLRenderSupported

Indicates whether a host or plug-in can support OpenGL accelerated rendering.

- Type - C string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only) - plug-in instance change (read/write)
- Default - "false" for a plug-in
- Valid Values - This must be one of
 - "false" - in which case the host or plug-in does not support OpenGL accelerated rendering
 - "true" - which means a host or plug-in can support OpenGL accelerated rendering, in the case of plug-ins this also means that it is capable of CPU based rendering in the absence of a GPU
 - "needed" - only for plug-ins, this means that an plug-in has to have OpenGL support, without which it cannot work.

V1.4: It is now expected from host reporting v1.4 that the plug-in can during instance change switch from true to false and false to true.

kOfxImageEffectPropOpenGLTextureIndex

Indicates the texture index of an image turned into an OpenGL texture by the host.

- Type - int X 1

- Property Set - texture handle returned by `OfxImageEffectOpenGLRenderSuiteV1::clipLoadTexture` (read only)

This value should be cast to a GLuint **and** used **as** the texture index when performing OpenGL texture operations.

The property set of the following actions should contain this property:

- `kOfxImageEffectActionRender`
- `kOfxImageEffectActionBeginSequenceRender`
- `kOfxImageEffectActionEndSequenceRender`

kOfxImageEffectPropOpenGLTextureTarget

Indicates the texture target enumerator of an image turned into an OpenGL texture by the host.

- Type - int X 1
- Property Set - texture handle returned by `OfxImageEffectOpenGLRenderSuiteV1::clipLoadTexture` (read only) This value should be cast to a GLenum and used as the texture target when performing OpenGL texture operations.

The property set of the following actions should contain this property:

- `kOfxImageEffectActionRender`
- `kOfxImageEffectActionBeginSequenceRender`
- `kOfxImageEffectActionEndSequenceRender`

kOfxImageEffectPropPixelFormat

Indicates the type of each component in a clip or image (after any mapping)

- Type - string X 1
- Property Set - clip instance (read only), image instance (read only)
- Valid Values - This must be one of
 - `kOfxBitDepthNone` (implying a clip is unconnected, not valid for an image)
 - `kOfxBitDepthByte`
 - `kOfxBitDepthShort`
 - `kOfxBitDepthHalf`
 - `kOfxBitDepthFloat`

Note that for a clip, this is the value set by the clip preferences action, not the raw ‘actual’ value of the clip.

kOfxImageEffectPropPluginHandle

The plugin handle passed to the initial ‘describe’ action.

- Type - pointer X 1
- Property Set - plugin instance, (read only)

This value will be the same for all instances of a plugin.

kOfxImageEffectPropPreMultiplication

Indicates the premultiplication state of a clip or image.

- Type - string X 1
- Property Set - clip instance (read only), image instance (read only), out args property in the *kOfxImageEffectActionGetClipPreferences* action (read/write)
- Valid Values - This must be one of
 - kOfxImageOpaque - the image is opaque and so has no premultiplication state
 - kOfxImagePreMultiplied - the image is premultiplied by its alpha
 - kOfxImageUnPreMultiplied - the image is unpremultiplied

See the documentation on clip preferences for more details on how this is used with the *kOfxImageEffectActionGetClipPreferences* action.

kOfxImageEffectPropProjectExtent

The extent of the current project in canonical coordinates.

- Type - double X 2
- Property Set - a plugin instance (read only)

The extent is the size of the ‘output’ for the current project. See *NormalisedCoordinateSystem* for more information on the project extent.

The extent is in canonical coordinates and only returns the top right position, as the extent is always rooted at 0,0.

For example a PAL SD project would have an extent of 768, 576.

kOfxImageEffectPropProjectOffset

The offset of the current project in canonical coordinates.

- Type - double X 2
- Property Set - a plugin instance (read only)

The offset is related to the *kOfxImageEffectPropProjectSize* and is the offset from the origin of the project ‘sub-window’.

For example for a PAL SD project that is in letterbox form, the project offset is the offset to the bottom left hand corner of the letter box.

The project offset is in canonical coordinates.

See *NormalisedCoordinateSystem* for more information on the project extent.

kOfxImageEffectPropProjectPixelAspectRatio

The pixel aspect ratio of the current project.

- Type - double X 1
- Property Set - a plugin instance (read only)

kOfxImageEffectPropProjectSize

The size of the current project in canonical coordinates.

- Type - double X 2
- Property Set - a plugin instance (read only)

The size of a project is a sub set of the *kOfxImageEffectPropProjectExtent*. For example a project may be a PAL SD project, but only be a letter-box within that. The project size is the size of this sub window.

The project size is in canonical coordinates.

See NormalisedCoordinateSystem for more information on the project extent.

kOfxImageEffectPropRegionOfDefinition

Used to indicate the region of definition of a plug-in.

- Type - double X 4
- Property Set - a read/write out argument property to the *kOfxImageEffectActionGetRegionOfDefinition* action
- Default - see *kOfxImageEffectActionGetRegionOfDefinition*

The order of the values is x1, y1, x2, y2.

This will be in CanonicalCoordinates

kOfxImageEffectPropRegionOfInterest

The value of a region of interest.

- Type - double X 4
- Property Set - a read only in argument property to the *kOfxImageEffectActionGetRegionsOfInterest* action

A host passes this value into the region of interest action to specify the region it is interested in rendering.

The order of the values is x1, y1, x2, y2.

This will be in CanonicalCoordinates.

kOfxImageEffectPropRenderQualityDraft

Indicates whether an effect can take quality shortcuts to improve speed.

- Type - int X 1
- Property Set - render calls, host (read-only)
- Default - 0 - 0: Best Quality (1: Draft)
- Valid Values - This must be one of 0 or 1

This property indicates that the host provides the plug-in the option to render in Draft/Preview mode. This is useful for applications that must support fast scrubbing. These allow a plug-in to take short-cuts for improved performance when the situation allows and it makes sense, for example to generate thumbnails with effects applied. For example switch to a cheaper interpolation type or rendering mode. A plugin should expect frames rendered in this manner that will not be stucked in host cache unless the cache is only used in the same draft situations. If an host does not support that property a value of 0 is assumed. Also note that some hosts do implement `kOfxImageEffectPropRenderScale` - these two properties can be used independently.

kOfxImageEffectPropRenderScale

The proxy render scale currently being applied.

- Type - double X 2
- Property Set - an image instance (read only) and as read only an in argument on the following actions,
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
 - *kOfxImageEffectActionIsIdentity*
 - *kOfxImageEffectActionGetRegionOfDefinition*
 - *kOfxImageEffectActionGetRegionsOfInterest*
 - *kOfxActionInstanceChanged*
 - *kOfxInteractActionDraw*
 - *kOfxInteractActionPenMotion*
 - *kOfxInteractActionPenDown*
 - *kOfxInteractActionPenUp*
 - *kOfxInteractActionKeyDown*
 - *kOfxInteractActionKeyUp*
 - *kOfxInteractActionKeyRepeat*
 - *kOfxInteractActionGainFocus*
 - *kOfxInteractActionLoseFocus*

This should be applied to any spatial parameters to position them correctly. Not that the 'x' value does not include any pixel aspect ratios.

kOfxImageEffectPropRenderWindow

The region to be rendered.

- Type - integer X 4
- Property Set - a read only in argument property to the *kOfxImageEffectActionRender* and *kOfxImageEffectActionIsIdentity* actions

The order of the values is x1, y1, x2, y2.

This will be in PixelCoordinates

kOfxImageEffectPropSequentialRenderStatus

Property on all the render action that indicate the current sequential render status of a host.

- Type - int X 1
- Property Set - read only property on the inArgs of the following actions...
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values -
 - 0 - the host is not currently sequentially rendering,
 - 1 - the host is currently rendering in a way so that it guarantees sequential rendering.

This property is set to indicate whether the effect is currently being rendered in frame order on a single effect instance. See *kOfxImageEffectInstancePropSequentialRender* for more details on sequential rendering.

kOfxImageEffectPropSettableFielding

Indicates whether the host will let a plugin set the fielding of the output clip.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 - in which case the plugin may not change the fielding of the output clip,
 - 1 - which means a plugin is able to change the output clip's fielding in the *kOfxImageEffectActionGetClipPreferences* action.

See ImageEffectClipPreferences.

kOfxImageEffectPropSettableFrameRate

Indicates whether the host will let a plugin set the frame rate of the output clip.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 - in which case the plugin may not change the frame rate of the output clip,
 - 1 - which means a plugin is able to change the output clip's frame rate in the *kOfxImageEffectActionGetClipPreferences* action.

See ImageEffectClipPreferences.

If a clip can be continuously sampled, the frame rate will be set to 0.

kOfxImageEffectPropSupportedComponents

Indicates the components supported by a clip or host,.

- Type - string X N
- Property Set - host descriptor (read only), clip descriptor (read/write)
- Valid Values - This must be one of
 - kOfxImageComponentNone (implying a clip is unconnected)
 - kOfxImageComponentRGBA
 - kOfxImageComponentRGB
 - kOfxImageComponentAlpha

This list of strings indicate what component types are supported by a host or are expected as input to a clip.

The default for a clip descriptor is to have none set, the plugin *must* define at least one in its define function

kOfxImageEffectPropSupportedContexts

Indicates to the host the contexts a plugin can be used in.

- Type - string X N
- Property Set - image effect descriptor passed to kOfxActionDescribe (read/write)
- Default - this has no defaults, it must be set
- Valid Values - This must be one of
 - *kOfxImageEffectContextGenerator*
 - *kOfxImageEffectContextFilter*
 - *kOfxImageEffectContextTransition*
 - *kOfxImageEffectContextPaint*
 - *kOfxImageEffectContextGeneral*
 - *kOfxImageEffectContextRetimer*

kOfxImageEffectPropSupportedPixelDepths

Indicates the bit depths support by a plug-in or host.

- Type - string X N
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - plugin descriptor none set
- Valid Values - This must be one of
 - kOfxBitDepthNone (implying a clip is unconnected, not valid for an image)
 - kOfxBitDepthByte
 - kOfxBitDepthShort

- kOfxBitDepthHalf
- kOfxBitDepthFloat

The default for a plugin is to have none set, the plugin *must* define at least one in its describe action.

kOfxImageEffectPropSupportsMultiResolution

Indicates whether a plugin or host support multiple resolution images.

- Type - int X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - 1 for plugins
- Valid Values - This must be one of
 - 0 - the plugin or host does not support multiple resolutions
 - 1 - the plugin or host does support multiple resolutions

Multiple resolution images mean...

- input and output images can be of any size
- input and output images can be offset from the origin

kOfxImageEffectPropSupportsMultipleClipDepths

Indicates whether a host or plugin can support clips of differing component depths going into/out of an effect.

- Type - int X 1
- Property Set - plugin descriptor (read/write), host descriptor (read only)
- Default - 0 for a plugin
- Valid Values - This must be one of
 - 0 - in which case the host or plugin does not support clips of multiple pixel depths,
 - 1 - which means a host or plugin is able to to deal with clips of multiple pixel depths,

If a host indicates that it can support multiple pixels depths, then it will allow the plugin to explicitly set the output clip's pixel depth in the *kOfxImageEffectActionGetClipPreferences* action. See *ImageEffectClipPreferences*.

kOfxImageEffectPropSupportsMultipleClipPARs

Indicates whether a host or plugin can support clips of differing pixel aspect ratios going into/out of an effect.

- Type - int X 1
- Property Set - plugin descriptor (read/write), host descriptor (read only)
- Default - 0 for a plugin
- Valid Values - This must be one of
 - 0 - in which case the host or plugin does not support clips of multiple pixel aspect ratios
 - 1 - which means a host or plugin is able to to deal with clips of multiple pixel aspect ratios

If a host indicates that it can support multiple pixel aspect ratios, then it will allow the plugin to explicitly set the output clip's aspect ratio in the *kOfxImageEffectActionGetClipPreferences* action. See *ImageEffectClipPreferences*.

kOfxImageEffectPropSupportsOverlays

Indicates whether a host support image effect *ImageEffectOverlays*.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 - the host won't allow a plugin to draw a GUI over the output image,
 - 1 - the host will allow a plugin to draw a GUI over the output image.

kOfxImageEffectPropSupportsTiles

Indicates whether a clip, plugin or host supports tiled images.

- Type - int X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write), clip descriptor (read/write), instance (read/write)
- Default - to 1 for a plugin and clip
- Valid Values - This must be one of 0 or 1

Tiled images mean that input or output images can contain pixel data that is only a subset of their full RoD.

If a clip or plugin does not support tiled images, then the host should supply full RoD images to the effect whenever it fetches one.

V1.4: It is now possible (defined) to change *OfxImageEffectPropSupportsTiles* in Instance Changed

kOfxImageEffectPropTemporalClipAccess

Indicates support for random temporal access to images in a clip.

- Type - int X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write), clip descriptor (read/write)
- Default - to 0 for a plugin and clip
- Valid Values - This must be one of 0 or 1

On a host, it indicates whether the host supports temporal access to images.

On a plugin, indicates if the plugin needs temporal access to images.

On a clip, it indicates that the clip needs temporal access to images.

kOfxImageEffectPropUnmappedFrameRange

The unmapped frame range over which an output clip has images.

- Type - double X 2
- Property Set - clip instance (read only)

Dimension 0 is the first frame for which the clip can produce valid data.

Dimension 1 is the last frame for which the clip can produce valid data.

If a plugin changes the output frame rate in the pixel preferences action, it will affect the frame range of the output clip, this property allows a plugin to get to the original value.

kOfxImageEffectPropUnmappedFrameRate

Indicates the original unmapped frame rate (frames/second) of a clip.

- Type - double X 1
- Property Set - clip instance (read only),

If a plugin changes the output frame rate in the pixel preferences action, this property allows a plugin to get to the original value.

kOfxImagePropBounds

The bounds of an image's pixels.

- Type - integer X 4
- Property Set - an image instance (read only)

The bounds, in PixelCoordinates, are of the addressable pixels in an image's data pointer.

The order of the values is x1, y1, x2, y2.

X values are $x1 \leq X < x2$ Y values are $y1 \leq Y < y2$

For less than full frame images, the pixel bounds will be contained by the *kOfxImagePropRegionOfDefinition* bounds.

kOfxImagePropData

The pixel data pointer of an image.

- Type - pointer X 1
- Property Set - an image instance (read only)

This property contains one of:

- a pointer to memory that is the lower left hand corner of an image
- a pointer to CUDA memory, if the Render action arguments includes *kOfxImageEffectPropCudaEnabled=1*
- an id<MTLBuffer>, if the Render action arguments includes *kOfxImageEffectPropMetalEnabled=1*
- a cl_mem, if the Render action arguments includes *kOfxImageEffectPropOpenCLEnabled=1*

See *kOfxImageEffectPropCudaEnabled*, *kOfxImageEffectPropMetalEnabled* and *kOfxImageEffectPropOpenCLEnabled*

kOfxImagePropField

Which fields are present in the image.

- Type - string X 1
- Property Set - an image instance (read only)
- Valid Values - This must be one of
 - *kOfxImageFieldNone* - the image is an unfielded frame
 - *kOfxImageFieldBoth* - the image is fielded and contains both interlaced fields
 - *kOfxImageFieldLower* - the image is fielded and contains a single field, being the lower field (rows 0,2,4...)
 - *kOfxImageFieldUpper* - the image is fielded and contains a single field, being the upper field (rows 1,3,5...)

kOfxImagePropPixelAspectRatio

The pixel aspect ratio of a clip or image.

- Type - double X 1
- Property Set - clip instance (read only), image instance (read only) and *kOfxImageEffectActionGetClipPreferences* action out args property (read/write)

kOfxImagePropRegionOfDefinition

The full region of definition of an image.

- Type - integer X 4
- Property Set - an image instance (read only)

An image's region of definition, in PixelCoordinates, is the full frame area of the image plane that the image covers.

The order of the values is x1, y1, x2, y2.

X values are $x1 \leq X < x2$ Y values are $y1 \leq Y < y2$

The *kOfxImagePropBounds* property contains the actual addressable pixels in an image, which may be less than its full region of definition.

kOfxImagePropRowBytes

The number of bytes in a row of an image.

- Type - integer X 1
- Property Set - an image instance (read only)

For various alignment reasons, a row of pixels may need to be padded at the end with several bytes before the next row starts in memory.

This property indicates the number of bytes in a row of pixels. This will be at least $\text{sizeof}(\text{PIXEL}) * (\text{bounds.x2} - \text{bounds.x1})$. Where *bounds* is fetched from the *kOfxImagePropBounds* property.

Note that (for CPU images only, not CUDA/Metal/OpenCL Buffers, nor OpenGL textures accessed via the OpenGL Render Suite) row bytes can be negative, which allows hosts with a native top down row order to pass image into OFX without having to repack pixels. Row bytes is not supported for OpenCL Images.

kOfxImagePropUniqueIdentifier

Uniquely labels an image.

- Type - ASCII string X 1
- Property Set - image instance (read only)

This is host set and allows a plug-in to differentiate between images. This is especially useful if a plugin caches analysed information about the image (for example motion vectors). The plugin can label the cached information with this identifier. If a user connects a different clip to the analysed input, or the image has changed in some way then the plugin can detect this via an identifier change and re-evaluate the cached information.

kOfxInteractPropBackgroundColour

The background colour of the application behind an interact instance.

- Type - double X 3
- Property Set - read only on the interact instance and in argument to the *kOfxInteractActionDraw* action
- Valid Values - from 0 to 1

The components are in the order red, green then blue.

kOfxInteractPropBitDepth

Indicates whether the dits per component in the interact's openGL frame buffer.

- Type - int X 1
- Property Set - interact instance and descriptor (read only)

kOfxInteractPropDrawContext

The Draw Context handle.

- Type - pointer X 1
- Property Set - read only property on the inArgs of the following actions...
- *kOfxInteractActionDraw*

kOfxInteractPropHasAlpha

Indicates whether the interact's frame buffer has an alpha component or not.

- Type - int X 1

- Property Set - interact instance and descriptor (read only)
- Valid Values - This must be one of
 - 0 indicates no alpha component
 - 1 indicates an alpha component

kOfxInteractPropPenPosition

The position of the pen in an interact.

- Type - double X 2
- Property Set - read only in argument to the *kOfxInteractActionPenMotion*, *kOfxInteractActionPenDown* and *kOfxInteractActionPenUp* actions

This value passes the position of the pen into an interact. This is in the interact's canonical coordinates.

kOfxInteractPropPenPressure

The pressure of the pen in an interact.

- Type - double X 1
- Property Set - read only in argument to the *kOfxInteractActionPenMotion*, *kOfxInteractActionPenDown* and *kOfxInteractActionPenUp* actions
- Valid Values - from 0 (no pressure) to 1 (maximum pressure)

This is used to indicate the status of the 'pen' in an interact. If a pen has only two states (eg: a mouse button), these should map to 0.0 and 1.0.

kOfxInteractPropPenViewportPosition

The position of the pen in an interact in viewport coordinates.

- Type - int X 2
- Property Set - read only in argument to the *kOfxInteractActionPenMotion*, *kOfxInteractActionPenDown* and *kOfxInteractActionPenUp* actions

This value passes the position of the pen into an interact. This is in the interact's OpenGL viewport coordinates, with 0,0 being at the bottom left.

kOfxInteractPropPixelScale

The size of a real screen pixel under the interact's canonical projection.

- Type - double X 2
- Property Set - interact instance and actions (read only)

kOfxInteractPropSlaveToParam

The set of parameters on which a value change will trigger a redraw for an interact.

- Type - string X N
- Property Set - interact instance property (read/write)
- Default - no values set
- Valid Values - the name of any parameter associated with this interact.

If the interact is representing the state of some set of OFX parameters, then it will need to be redrawn if any of those parameters' values change. This multi-dimensional property links such parameters to the interact.

The interact can be slaved to multiple parameters (setting index 0, then index 1 etc...)

kOfxInteractPropSuggestedColour

The suggested colour to draw a widget in an interact, typically for overlays.

- Type - double X 3
- Property Set - read only on the interact instance
- Default - 1.0
- Valid Values - greater than or equal to 0.0

Some applications allow the user to specify colours of any overlay via a colour picker, this property represents the value of that colour. Plugins are at liberty to use this or not when they draw an overlay.

If a host does not support such a colour, it should return `kOfxStatReplyDefault`

kOfxInteractPropViewportSize

The size of an interact's OpenGL viewport [#8212; *ofxInteract.h*](#).

- Type - int X 2
- Property Set - read only property on the interact instance and in argument to all the interact actions.

Deprecated:

- V1.3: This property is the redundant and its use will be deprecated in future releases. V1.4: Removed

kOfxOpenGLPropPixelDepth

Indicates the bit depths supported by a plug-in during OpenGL renders.

This is analogous to *kOfxImageEffectPropSupportedPixelDepths*. When a plug-in sets this property, the host will try to provide buffers/textures in one of the supported formats. Additionally, the target buffers where the plug-in renders to will be set to one of the supported formats.

Unlike *kOfxImageEffectPropSupportedPixelDepths*, this property is optional. Shader-based effects might not really care about any format specifics when using OpenGL textures, so they can leave this unset and allow the host to decide the format.

- Type - string X N
- Property Set - plug-in descriptor (read only)
- Default - none set

- Valid Values - This must be one of
 - *kOfxBitDepthNone* (implying a clip is unconnected, not valid for an image)
 - *kOfxBitDepthByte*
 - *kOfxBitDepthShort*
 - *kOfxBitDepthHalf*
 - *kOfxBitDepthFloat*

kOfxParamHostPropMaxPages

Indicates the maximum number of parameter pages.

- Type - int X 1
- Property Set - host descriptor (read only)

If there is no limit to the number of pages on a host, set this to -1.

Hosts that do not support paged parameter layout should set this to zero.

kOfxParamHostPropMaxParameters

Indicates the maximum numbers of parameters available on the host.

- Type - int X 1
- Property Set - host descriptor (read only)

If set to -1 it implies unlimited number of parameters.

kOfxParamHostPropPageRowColumnCount

This indicates the number of parameter rows and columns on a page.

- Type - int X 2
- Property Set - host descriptor (read only)

If the host has supports paged parameter layout, used dimension 0 as the number of columns per page and dimension 1 as the number of rows per page.

kOfxParamHostPropSupportsBooleanAnimation

Indicates if the host supports animation of boolean params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

kOfxParamHostPropSupportsChoiceAnimation

Indicates if the host supports animation of choice params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

kOfxParamHostPropSupportsCustomAnimation

Indicates if the host supports animation of custom parameters.

- Type - int X 1
- Property Set - host descriptor (read only)
- Value Values - 0 or 1

kOfxParamHostPropSupportsCustomInteract

Indicates if the host supports custom interacts for parameters.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

Currently custom interacts for parameters can only be drawn using OpenGL. APIs will be added later to support using the new Draw Suite.

kOfxParamHostPropSupportsParametricAnimation

Property on the host to indicate support for parametric parameter animation.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values
 - 0 indicating the host does not support animation of parametric params,
 - 1 indicating the host does support animation of parametric params,

kOfxParamHostPropSupportsStrChoice

Indicates if the host supports the StrChoice param type.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

Since
Version 1.5

kOfxParamHostPropSupportsStrChoiceAnimation

Indicates if the host supports animation of string choice params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

Since

Version 1.5

kOfxParamHostPropSupportsStringAnimation

Indicates if the host supports animation of string params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

kOfxParamPropAnimates

Flags whether a parameter can animate.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

A plug-in uses this property to indicate if a parameter is able to animate.

kOfxParamPropCacheInvalidation

Specifies how modifying the value of a param will affect any output of an effect over time.

- Type - C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - *kOfxParamInvalidateValueChange*
- Valid Values - This must be one of
 - *kOfxParamInvalidateValueChange*
 - *kOfxParamInvalidateValueChangeToEnd*
 - *kOfxParamInvalidateAll*

Imagine an effect with an animating parameter in a host that caches rendered output. Think of the what happens when you add a new key frame. -If the parameter represents something like an absolute position, the cache will only need to be invalidated for the range of frames that keyframe affects.

- If the parameter represents something like a speed which is integrated, the cache will be invalidated from the keyframe until the end of the clip.
- There are potentially other situations where the entire cache will need to be invalidated (though I can't think of one off the top of my head).

kOfxParamPropCanUndo

Flags whether changes to a parameter should be put on the undo/redo stack.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

kOfxParamPropChoiceEnum

Set a enumeration string in a StrChoice (string-valued choice) parameter.

- Type - UTF8 C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the property is empty with no options set.

This property contains the set of enumeration strings stored by the host in the project corresponding to the options that will be presented to a user from a StrChoice parameter. See ParametersChoice for more details.

Since

Version 1.5

kOfxParamPropChoiceOption

Set options of a choice parameter.

- Type - UTF8 C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the property is empty with no options set.

This property contains the set of options that will be presented to a user from a choice parameter. See ParametersChoice for more details.

kOfxParamPropChoiceOrder

Set values the host should store for a choice parameter.

- Type - int X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - Zero-based ordinal list of same length as OfxParamPropChoiceOption

This property specifies the order in which the options are presented. See “Choice Parameters” for more details. This property is optional; if not set, the host will present the options in their natural order.

This property is useful when changing order of choice param options, or adding new options in the middle, in a new version of the plugin.

```

Plugin v1:
Option = {"OptA", "OptB", "OptC"}
Order = {1, 2, 3}

Plugin v2:
// will be shown as OptA / OptB / NewOpt / OptC
Option = {"OptA", "OptB", "OptC", NewOpt"}
Order = {1, 2, 4, 3}

```

Note that this only affects the host UI’s display order; the project still stores the index of the selected option as always. Plugins should never reorder existing options if they desire backward compatibility.

Values may be arbitrary 32-bit integers. Behavior is undefined if the same value occurs twice in the list; plugins should not do that.

Since

Version 1.5

kOfxParamPropCustomInterpCallbackV1

A pointer to a custom parameter’s interpolation function.

- Type - pointer X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - NULL
- Valid Values - must point to a *OfxCustomParamInterpFuncV1*

It is an error not to set this property in a custom parameter during a plugin’s define call if the custom parameter declares itself to be an animating parameter.

kOfxParamPropCustomValue

Used by interpolating custom parameters to get and set interpolated values.

- Type - C string X 1 or 2

This property is on the *inArgs* property and *outArgs* property of a *OfxCustomParamInterpFuncV1* and in both cases contains the encoded value of a custom parameter. As an *inArgs* property it will have two values, being the two keyframes to interpolate. As an *outArgs* property it will have a single value and the plugin should fill this with the encoded interpolated value of the parameter.

kOfxParamPropDataPtr

A private data pointer that the plug-in can store its own data behind.

- Type - pointer X 1

- Property Set - plugin parameter instance (read/write),
- Default - NULL

This data pointer is unique to each parameter instance, so two instances of the same parameter do not share the same data pointer. Use it to hang any needed private data structures.

kOfxParamPropDefault

The default value of a parameter.

- Type - The type is dependant on the parameter type as is the dimension.
- Property Set - plugin parameter descriptor (read/write) and instance (read/write only),
- Default - 0 cast to the relevant type (or "" for strings and custom parameters)

The exact type and dimension is dependant on the type of the parameter. These are...

- *kOfxParamTypeInteger* - integer property of one dimension
- *kOfxParamTypeDouble* - double property of one dimension
- *kOfxParamTypeBoolean* - integer property of one dimension
- *kOfxParamTypeChoice* - integer property of one dimension
- *kOfxParamTypeStrChoice* - string property of one dimension
- *kOfxParamTypeRGBA* - double property of four dimensions
- *kOfxParamTypeRGB* - double property of three dimensions
- *kOfxParamTypeDouble2D* - double property of two dimensions
- *kOfxParamTypeInteger2D* - integer property of two dimensions
- *kOfxParamTypeDouble3D* - double property of three dimensions
- *kOfxParamTypeInteger3D* - integer property of three dimensions
- *kOfxParamTypeString* - string property of one dimension
- *kOfxParamTypeCustom* - string property of one dimension
- *kOfxParamTypeGroup* - does not have this property
- *kOfxParamTypePage* - does not have this property
- *kOfxParamTypePushButton* - does not have this property

kOfxParamPropDefaultCoordinateSystem

Describes in which coordinate system a spatial double parameter's default value is specified.

- Type - C string X 1
- Default - kOfxParamCoordinatesCanonical
- Property Set - Non normalised spatial double parameters, ie: any double param who's *kOfxParamProp-DoubleType* is set to one of...
 - kOfxParamDoubleTypeX
 - kOfxParamDoubleTypeXAbsolute

- kOfxParamDoubleTypeY
- kOfxParamDoubleTypeYAbsolute
- kOfxParamDoubleTypeXY
- kOfxParamDoubleTypeXYAbsolute
- Valid Values - This must be one of
 - kOfxParamCoordinatesCanonical - the default is in canonical coords
 - kOfxParamCoordinatesNormalised - the default is in normalised coordinates

This allows a spatial param to specify what its default is, so by saying normalised and “0.5” it would be in the ‘middle’, by saying canonical and 100 it would be at value 100 independent of the size of the image being applied to.

kOfxParamPropDigits

How many digits after a decimal point to display for a double param in a GUI.

- Type - int X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - 2
- Valid Values - any greater than 0.

This applies to double params of any dimension.

kOfxParamPropDimensionLabel

Label for individual dimensions on a multidimensional numeric parameter.

- Type - UTF8 C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - “x”, “y” and “z”
- Valid Values - any

Use this on 2D and 3D double and integer parameters to change the label on an individual dimension in any GUI for that parameter.

kOfxParamPropDisplayMax

The maximum value for a numeric parameter on any user interface.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the largest possible value corresponding to the parameter type (eg: INT_MAX for an integer, DBL_MAX for a double parameter)

If a user interface represents a parameter with a slider or similar, this should be the maximum bound on that slider.

kOfxParamPropDisplayMin

The minimum value for a numeric parameter on any user interface.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the smallest possible value corresponding to the parameter type (eg: INT_MIN for an integer, -DBL_MAX for a double parameter)

If a user interface represents a parameter with a slider or similar, this should be the minimum bound on that slider.

kOfxParamPropDoubleType

Describes how the double parameter should be interpreted by a host.

- Type - C string X 1
- Default - *kOfxParamDoubleTypePlain*
- Property Set - 1D, 2D and 3D float plugin parameter descriptor (read/write) and instance (read only),
- Valid Values - This must be one of
 - *kOfxParamDoubleTypePlain* - parameter has no special interpretation,
 - *kOfxParamDoubleTypeAngle* - parameter is to be interpreted as an angle,
 - *kOfxParamDoubleTypeScale* - parameter is to be interpreted as a scale factor,
 - *kOfxParamDoubleTypeTime* - parameter represents a time value (1D only),
 - *kOfxParamDoubleTypeAbsoluteTime* - parameter represents an absolute time value (1D only),
 - *kOfxParamDoubleTypeX* - size wrt to the project's X dimension (1D only), in canonical coordinates,
 - *kOfxParamDoubleTypeXAbsolute* - absolute position on the X axis (1D only), in canonical coordinates,
 - *kOfxParamDoubleTypeY* - size wrt to the project's Y dimension (1D only), in canonical coordinates,
 - *kOfxParamDoubleTypeYAbsolute* - absolute position on the Y axis (1D only), in canonical coordinates,
 - *kOfxParamDoubleTypeXY* - size in 2D (2D only), in canonical coordinates,
 - *kOfxParamDoubleTypeXYAbsolute* - an absolute position on the image plane, in canonical coordinates.

Double parameters can be interpreted in several different ways, this property tells the host how to do so and thus gives hints as to the interface of the parameter.

kOfxParamPropEnabled

Used to enable a parameter in the user interface.

- Type - int X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - 1
- Valid Values - 0 or 1

When set to 0 a user should not be able to modify the value of the parameter. Note that the plug-in itself can still change the value of a disabled parameter.

kOfxParamPropEvaluateOnChange

Flags whether changing a parameter's value forces an evaluation (ie: render),.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write only)
- Default - 1
- Valid Values - 0 or 1

This is used to indicate if the value of a parameter has any affect on an effect's output, eg: the parameter may be purely for GUI purposes, and so changing its value should not trigger a re-render.

kOfxParamPropGroupOpen

Whether the initial state of a group is open or closed in a hierarchical layout.

- Type - int X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

This is a property on parameters of type *kOfxParamTypeGroup*, and tells the group whether it should be open or closed by default.

kOfxParamPropHasHostOverlayHandle

A flag to indicate if there is a host overlay UI handle for the given parameter.

- Type - int x 1
- Property Set - plugin parameter descriptor (read only)
- Valid Values - 0 or 1

If set to 1, then the host is flagging that there is some sort of native user overlay interface handle available for the given parameter.

kOfxParamPropHint

A hint to the user as to how the parameter is to be used.

- Type - UTF8 C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - ""

kOfxParamPropIncrement

The granularity of a slider used to represent a numeric parameter.

- Type - double X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - 1
- Valid Values - any greater than 0.

This value is always in canonical coordinates for double parameters that are normalised.

kOfxParamPropInteractMinimumSize

The minimum size of a parameter's custom interface, in screen pixels.

- Type - double x 2
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 10,10
- Valid Values - greater than (0, 0)

Any custom interface will not be less than this size.

kOfxParamPropInteractPreferredSize

The preferred size of a parameter's custom interface.

- Type - int x 2
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 10,10
- Valid Values - greater than (0, 0)

A host should attempt to set a parameter's custom interface on a parameter to be this size if possible, otherwise it will be of *kOfxParamPropInteractSizeAspect* aspect but larger than *kOfxParamPropInteractMinimumSize*.

kOfxParamPropInteractSize

The size of a parameter instance's custom interface in screen pixels.

- Type - double x 2
- Property Set - plugin parameter instance (read only)

This is set by a host to indicate the current size of a custom interface if the plug-in has one. If not this is set to (0,0).

kOfxParamPropInteractSizeAspect

The preferred aspect ratio of a parameter's custom interface.

- Type - double x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1.0
- Valid Values - greater than or equal to 0.0

If set to anything other than 0.0, the custom interface for this parameter will be of a size with this aspect ratio (x size/y size).

kOfxParamPropInteractV1

Overrides the parameter's standard user interface with the given interact.

- Type - pointer X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - NULL
- Valid Values - must point to a *OfxPluginEntryPoint*

If set, the parameter's normal interface is replaced completely by the interact gui.

Currently custom interacts for parameters can only be drawn using OpenGL. APIs will be added later to support using the new Draw Suite.

kOfxParamPropInterpolationAmount

Property used by *OfxCustomParamInterpFuncV1* to indicate the amount of interpolation to perform.

- Type - double X 1
- Property Set - inArgs parameter of a *OfxCustomParamInterpFuncV1* (read only)
- Valid Values - from 0 to 1

This property indicates how far between the two *kOfxParamPropCustomValue* keys to interpolate.

kOfxParamPropInterpolationTime

Used by interpolating custom parameters to indicate the time a key occurs at.

- Type - double X 2
- Property Set - inArgs parameter of a *OfxCustomParamInterpFuncV1* (read only)

The two values indicate the absolute times the surrounding keyframes occur at. The keyframes are encoded in a *kOfxParamPropCustomValue* property.

kOfxParamPropIsAnimating

Flags whether a parameter is currently animating.

- Type - int x 1
- Property Set - plugin parameter instance (read only)
- Valid Values - 0 or 1

Set by a host on a parameter instance to indicate if the parameter has a non-constant value set on it. This can be as a consequence of animation or of scripting modifying the value, or of a parameter being connected to an expression in the host.

kOfxParamPropIsAutoKeying

Will a value change on the parameter add automatic keyframes.

- Type - int X 1
- Property Set - plugin parameter instance (read only),
- Valid Values - 0 or 1

This is set by the host simply to indicate the state of the property.

kOfxParamPropMax

The maximum value for a numeric parameter.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the largest possible value corresponding to the parameter type (eg: INT_MAX for an integer, DBL_MAX for a double parameter)

Setting this will also reset `;;kOfxParamPropDisplayMax`.

kOfxParamPropMin

The minimum value for a numeric parameter.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the smallest possible value corresponding to the parameter type (eg: INT_MIN for an integer, -DBL_MAX for a double parameter)

Setting this will also reset `kOfxParamPropDisplayMin`.

kOfxParamPropPageChild

The names of the parameters included in a page parameter.

- Type - C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - ""
- Valid Values - the names of any parameter that is not a group or page, as well as `kOfxParamPageSkipRow` and `kOfxParamPageSkipColumn`

This is a property on parameters of type `kOfxParamTypePage`, and tells the page what parameters it contains. The parameters are added to the page from the top left, filling in columns as we go. The two pseudo param names `kOfxParamPageSkipRow` and `kOfxParamPageSkipColumn` are used to control layout.

Note parameters can appear in more than one page.

kOfxParamPropParametricDimension

The dimension of a parametric param.

- Type - int X 1
- Property Set - parametric param descriptor (read/write) and instance (read only)
- default - 1
- Value Values - greater than 0

This indicates the dimension of the parametric param.

kOfxParamPropParametricInteractBackground

Interact entry point to draw the background of a parametric parameter.

- Type - pointer X 1
- Property Set - plug-in parametric parameter descriptor (read/write) and instance (read only),
- Default - NULL, which implies the host should draw its default background.

Defines a pointer to an interact which will be used to draw the background of a parametric parameter's user interface. None of the pen or keyboard actions can ever be called on the interact.

The OpenGL transform will be set so that it is an orthographic transform that maps directly to the 'parametric' space, so that 'x' represents the parametric position and 'y' represents the evaluated value.

kOfxParamPropParametricRange

Property to indicate the min and max range of the parametric input value.

- Type - double X 2
- Property Set - parameter descriptor (read/write only), and instance (read only)
- Default Value - (0, 1)
- Valid Values - any pair of numbers so that the first is less than the second.

This controls the min and max values that the parameter will be evaluated at.

kOfxParamPropParametricUIColor

The colour of parametric param curve interface in any UI.

- Type - double X N
- Property Set - parametric param descriptor (read/write) and instance (read only)
- default - unset,
- Value Values - three values for each dimension (see *kOfxParamPropParametricDimension*) being interpreted as R, G and B of the colour for each curve drawn in the UI.

This sets the colour of a parametric param curve drawn a host user interface. A colour triple is needed for each dimension of the parametric param.

If not set, the host should generally draw these in white.

kOfxParamPropParent

The name of a parameter's parent group.

- Type - C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - "", which implies the "root" of the hierarchy,
- Valid Values - the name of a parameter with type of *kOfxParamTypeGroup*

Hosts that have hierarchical layouts of their params use this to recursively group parameter.

By default parameters are added in order of declaration to the 'root' hierarchy. This property is used to reparam params to a predefined param of type *kOfxParamTypeGroup*.

kOfxParamPropPersistent

Flags whether the value of a parameter should persist.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

This is used to tell the host whether the value of the parameter is important and should be save in any description of the plug-in.

kOfxParamPropPluginMayWrite

Flags whether the plugin will attempt to set the value of a parameter in some callback or analysis pass.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 0
- Valid Values - 0 or 1

This is used to tell the host whether the plug-in is going to attempt to set the value of the parameter.

Deprecated:

- v1.4: deprecated - to be removed in 1.5

kOfxParamPropScriptName

The value to be used as the id of the parameter in a host scripting language.

- Type - ASCII C string X 1,
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - the unique name the parameter was created with.
- Valid Values - ASCII string unique to all parameters in the plug-in.

Many hosts have a scripting language that they use to set values of parameters and more. If so, this is the name of a parameter in such scripts.

kOfxParamPropSecret

Flags whether a parameter should be exposed to a user,.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write)
- Default - 0
- Valid Values - 0 or 1

If secret, a parameter is not exposed to a user in any interface, but should otherwise behave as a normal parameter.

Secret params are typically used to hide important state detail that would otherwise be unintelligible to a user, for example the result of a statical analysis that might need many parameters to store.

kOfxParamPropShowTimeMarker

Enables the display of a time marker on the host's time line to indicate the value of the absolute time param.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write)
- Default - 0
- Valid Values - 0 or 1

If a double parameter is has *kOfxParamPropDoubleType* set to *kOfxParamDoubleTypeAbsoluteTime*, then this indicates whether any marker should be made visible on the host's time line.

kOfxParamPropStringMode

Used to indicate the type of a string parameter.

- Type - C string X 1
- Property Set - plugin string parameter descriptor (read/write) and instance (read only),
- Default - *kOfxParamStringIsSingleLine*
- Valid Values - This must be one of the following
 - *kOfxParamStringIsSingleLine*
 - *kOfxParamStringIsMultiLine*
 - *kOfxParamStringIsFilePath*
 - *kOfxParamStringIsDirectoryPath*

- *kOfxParamStringIsLabel*
- *kOfxParamStringIsRichTextFormat*

kOfxParamPropType

The type of a parameter.

- Type - C string X 1
- Property Set - plugin parameter descriptor (read only) and instance (read only)

This string will be set to the type that the parameter was create with.

kOfxParamPropUseHostOverlayHandle

A flag to indicate that the host should use a native UI overlay handle for the given parameter.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write only) and instance (read only)
- Default - 0
- Valid Values - 0 or 1

If set to 1, then a plugin is flaging to the host that the host should use a native UI overlay handle for the given parameter. A plugin can use this to keep a native look and feel for parameter handles. A plugin can use *kOfxParamPropHasHostOverlayHandle* to see if handles are available on the given parameter.

kOfxPluginPropFilePath

The file path to the plugin.

- Type - C string X 1
- Property Set - effect descriptor (read only)

This is a string that indicates the file path where the plug-in was found by the host. The path is in the native path format for the host OS (eg: UNIX directory separators are forward slashes, Windows ones are backslashes).

The path is to the bundle location, see *InstallationLocation*. eg: `‘/usr/OFX/Plugins/AcmePlugins/AcmeFantasticPlugin.ofx.bundle’`

kOfxPluginPropParamPageOrder

Sets the parameter pages and order of pages.

- Type - C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - “”
- Valid Values - the names of any page param in the plugin

This property sets the preferred order of parameter pages on a host. If this is never set, the preferred order is the order the parameters were declared in.

kOfxPropAPIVersion

Property on the host descriptor, saying what API version of the API is being implemented.

- Type - int X N
- Property Set - host descriptor.

This is a version string that will specify which version of the API is being implemented by a host. It can have multiple values. For example “1.0”, “1.2.4” etc....

If this is not present, it is safe to assume that the version of the API is “1.0”.

kOfxPropChangeReason

Indicates why a plug-in changed.

- Type - ASCII C string X 1
- Property Set - inArgs parameter on the *kOfxActionInstanceChanged* action.
- Valid Values - this can be...
 - *kOfxChangeUserEdited* - the user directly edited the instance somehow and caused a change to something, this includes undo/redos and resets
 - *kOfxChangePluginEdited* - the plug-in itself has changed the value of the object in some action
 - *kOfxChangeTime* - the time has changed and this has affected the value of the object because it varies over time

Argument property for the *kOfxActionInstanceChanged* action.

kOfxPropEffectInstance

A pointer to an effect instance.

- Type - pointer X 1
- Property Set - on an interact instance (read only)

This property is used to link an object to the effect. For example if the plug-in supplies an OpenGL overlay for an image effect, the interact instance will have one of these so that the plug-in can connect back to the effect the GUI links to.

kOfxPropHostOSHandle

A pointer to an operating system specific application handle.

- Type - pointer X 1
- Property Set - host descriptor.

Some plug-in vendor want raw OS specific handles back from the host so they can do interesting things with host OS APIs. Typically this is to control windowing properly on Microsoft Windows. This property returns the appropriate ‘root’ window handle on the current operating system. So on Windows this would be the hWnd of the application main window.

kOfxPropIcon

If set this tells the host to use an icon instead of a label for some object in the interface.

- Type - string X 2
- Property Set - various descriptors in the API
- Default - ""
- Valid Values - ASCII string

The value is a path is defined relative to the Resource folder that points to an SVG or PNG file containing the icon.

The first dimension, if set, will the name of and SVG file, the second a PNG file.

kOfxPropInstanceData

A private data pointer that the plug-in can store its own data behind.

- Type - pointer X 1
- Property Set - plugin instance (read/write),
- Default - NULL

This data pointer is unique to each plug-in instance, so two instances of the same plug-in do not share the same data pointer. Use it to hang any needed private data structures.

kOfxPropIsInteractive

Indicates if a host is actively editing the effect with some GUI.

- Type - int X 1
- Property Set - effect instance (read only)
- Valid Values - 0 or 1

If false the effect currently has no interface, however this may be because the effect is loaded in a background render host, or it may be loaded on an interactive host that has not yet opened an editor for the effect.

The output of an effect should only ever depend on the state of its parameters, not on the interactive flag. The interactive flag is more a courtesy flag to let a plugin know that it has an interace. If a plugin want's to have its behaviour dependant on the interactive flag, it can always make a secret parameter which shadows the state if the flag.

kOfxPropKeyString

This property encodes a single keypresses that generates a unicode code point. The value is stored as a UTF8 string.

- Type - C string X 1, UTF8
- Property Set - an read only in argument for the actions *kOfxInteractActionKeyDown*, *kOfxInteractActionKeyUp* and *kOfxInteractActionKeyRepeat*.

- Valid Values - a UTF8 string representing a single character, or the empty string.

This property represents the UTF8 encode value of a single key press by a user in an OFX interact.

This property is associated with a *kOfxPropKeySym* which represents an integer value for the key press. Some keys (for example arrow keys) have no UTF8 equivalent, in which case this is set to the empty string "", and the associate *kOfxPropKeySym* is set to the equivalent raw key press.

Some keys, especially on non-english language systems, may have a UTF8 value, but *not* a keysym values, in these cases, the keysym will have a value of `kOfxKey_Unknown`, but the *kOfxPropKeyString* property will still be set with the UTF8 value.

kOfxPropKeySym

Property used to indicate which a key on the keyboard or a button on a button device has been pressed.

- Type - int X 1
- Property Set - an read only in argument for the actions *kOfxInteractActionKeyDown*, *kOfxInteractActionKeyUp* and *kOfxInteractActionKeyRepeat*.
- Valid Values - one of any specified by #defines in the file *ofxKeySyms.h*.

This property represents a raw key press, it does not represent the 'character value' of the key.

This property is associated with a *kOfxPropKeyString* property, which encodes the UTF8 value for the key-press/button press. Some keys (for example arrow keys) have no UTF8 equivalent.

Some keys, especially on non-english language systems, may have a UTF8 value, but *not* a keysym values, in these cases, the keysym will have a value of `kOfxKey_Unknown`, but the *kOfxPropKeyString* property will still be set with the UTF8 value.

kOfxPropLabel

User visible name of an object.

- Type - UTF8 C string X 1
- Property Set - on many objects (descriptors and instances), see PropertiesByObject. Typically readable and writable in most cases.
- Default - the *kOfxPropName* the object was created with.

The label is what a user sees on any interface in place of the object's name.

Note that resetting this will also reset *kOfxPropShortLabel* and *kOfxPropLongLabel*.

kOfxPropLongLabel

Long user visible name of an object.

- Type - UTF8 C string X 1
- Property Set - on many objects (descriptors and instances), see PropertiesByObject. Typically readable and writable in most cases.
- Default - initially *kOfxPropName*, but will be reset if *kOfxPropLabel* is changed.

This is a longer version of the label, typically 32 character glyphs or so. Hosts should use this if they have much display space for their object labels.

kOfxPropName

Unique name of an object.

- Type - ASCII C string X 1
- Property Set - on many objects (descriptors and instances), see `PropertiesByObject` (read only)

This property is used to label objects uniquely among objects of that type. It is typically set when a plugin creates a new object with a function that takes a name.

kOfxPropParamSetNeedsSyncing

States whether the plugin needs to resync its private data.

- Type - int X 1
- Property Set - param set instance (read/write)
- Default - 0
- Valid Values -
 - 0 - no need to sync
 - 1 - paramset is not synced

The plugin should set this flag to true whenever any internal state has not been flushed to the set of params.

The host will examine this property each time it does a copy or save operation on the instance. If it is set to 1, the host will call `SyncPrivateData` and then set it to zero before doing the copy/save. If it is set to 0, the host will assume that the param data correctly represents the private state, and will not call `SyncPrivateData` before copying/saving. If this property is not set, the host will always call `SyncPrivateData` before copying or saving the effect (as if the property were set to 1 — but the host will not create or modify the property).

kOfxPropPluginDescription

Description of the plug-in to a user.

- Type - string X 1
- Property Set - plugin descriptor (read/write) and instance (read only)
- Default - ""
- Valid Values - UTF8 string

This is a string giving a potentially verbose description of the effect.

kOfxPropShortLabel

Short user visible name of an object.

- Type - UTF8 C string X 1
- Property Set - on many objects (descriptors and instances), see `PropertiesByObject`. Typically readable and writable in most cases.
- Default - initially *kOfxPropName*, but will be reset if *kOfxPropLabel* is changed.

This is a shorter version of the label, typically 13 character glyphs or less. Hosts should use this if they have limited display space for their object labels.

kOfxPropTime

General property used to get/set the time of something.

- Type - double X 1
- Default - 0, if a settable property
- Property Set - commonly used as an argument to actions, input and output.

kOfxPropType

General property, used to identify the kind of an object behind a handle.

- Type - ASCII C string X 1
- Property Set - any object handle (read only)
- Valid Values - currently this can be...
 - *kOfxTypeImageEffectHost*
 - *kOfxTypeImageEffect*
 - *kOfxTypeImageEffectInstance*
 - *kOfxTypeParameter*
 - *kOfxTypeParameterInstance*
 - *kOfxTypeClip*
 - *kOfxTypeImage*

kOfxPropVersion

Identifies a specific version of a host or plugin.

- Type - int X N
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - “0”
- Valid Values - positive integers

This is a multi dimensional integer property that represents the version of a host (host descriptor), or plugin (plugin descriptor). These represent a version number of the form ‘1.2.3.4’, with each dimension adding another ‘dot’ on the right.

A version is considered to be more recent than another if its ordered set of values is lexicographically greater than another, reading left to right. (ie: 1.2.4 is smaller than 1.2.6). Also, if the number of dimensions is different, then the values of the missing dimensions are considered to be zero (so 1.2.4 is greater than 1.2).

kOfxPropVersionLabel

Unique user readable version string of a plugin or host.

- Type - string X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - none, the host needs to set this
- Valid Values - ASCII string

This is purely for user feedback, a plugin or host should use *kOfxPropVersion* if they need to check for specific versions.

1.21 Auto-generated Reference Index

1.21.1 File list

File ofxCORE.h

Contains the core OFX architectural struct and function definitions. For more details on the basic OFX architecture, see Architecture.

Defines

OfxExport

Platform independent export macro.

This macro is to be used before any symbol that is to be exported from a plug-in. This is OS/compiler dependent.

kOfxActionLoad

This action is the first action passed to a plug-in after the binary containing the plug-in has been loaded. It is there to allow a plug-in to create any global data structures it may need and is also when the plug-in should fetch suites from the host.

The handle, inArgs and outArgs arguments to the mainEntry are redundant and should be set to NULL.

Pre

- The plugin's *OfxPlugin::setHost* function has been called

Post

This action will not be called again while the binary containing the plug-in remains loaded.

Returns

- *kOfxStatOK*, the action was trapped and all was well,
- *kOfxStatReplyDefault*, the action was ignored,
- *kOfxStatFailed*, the load action failed, no further actions will be passed to the plug-in. Interpret if possible kOfxStatFailed as plug-in indicating it does not want to load Do not create an entry in the host's UI for plug-in then.

Plug-in also has the option to return 0 for `OfxGetNumberOfPlugins` or `kOfxStatFailed` if host supports `OfxSetHost` in which case `kOfxActionLoad` will never be called.

- *kOfxStatErrFatal*, fatal error in the plug-in.

kOfxActionDescribe

The `kOfxActionDescribe` is the second action passed to a plug-in. It is where a plugin defines how it behaves and the resources it needs to function.

Note that the handle passed in acts as a descriptor for, rather than an instance of the plugin. The handle is global and unique. The plug-in is at liberty to cache the handle away for future reference until the plug-in is unloaded.

Most importantly, the effect must set what image effect contexts it is capable of working in.

This action *must* be trapped, it is not optional.

Parameters

- **handle** – handle to the plug-in descriptor, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionLoad* has been called

Post

- *kOfxActionDescribe* will not be called again, unless it fails and returns one of the error codes where the host is allowed to attempt the action again
- the handle argument, being the global plug-in description handle, is a valid handle from the end of a successful describe action until the end of the *kOfxActionUnload* action (ie: the plug-in can cache it away without worrying about it changing between actions).
- *kOfxImageEffectActionDescribeInContext* will be called once for each context that the host and plug-in mutually support. If a plug-in does not report to support any context supported by host, host should not enable the plug-in.

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatErrMissingHostFeature*, in which the plugin will be unloaded and ignored, plugin may post message
- *kOfxStatErrMemory*, in which case describe may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxActionUnload

This action is the last action passed to the plug-in before the binary containing the plug-in is unloaded. It is there to allow a plug-in to destroy any global data structures it may have created.

The handle, inArgs and outArgs arguments to the main entry are redundant and should be set to NULL.

Pre

- the *kOfxActionLoad* action has been called
- all instances of a plugin have been destroyed

Post

- No other actions will be called.

Returns

- *kOfxStatOK*, the action was trapped all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*, in which case we the program will be forced to quit

kOfxActionPurgeCaches

This action is an action that may be passed to a plug-in instance from time to time in low memory situations. Instances receiving this action should destroy any data structures they may have and release the associated memory, they can later reconstruct this from the effect's parameter set and associated information.

For Image Effects, it is generally a bad idea to call this after each render, but rather it should be called after *kOfxImageEffectActionEndSequenceRender*. Some effects, typically those flagged with the *kOfxImageEffectInstancePropSequentialRender* property, may need to cache information from previously rendered frames to function correctly, or have data structures that are expensive to reconstruct at each frame (eg: a particle system). Ideally, such effect should free such structures during the *kOfxImageEffectActionEndSequenceRender* action.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionSyncPrivateData

This action is called when a plugin should synchronise any private data structures to its parameter set. This generally occurs when an effect is about to be saved or copied, but it could occur in other situations as well.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Post

- Any private state data can be reconstructed from the parameter set,

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionCreateInstance

This action is the first action passed to a plug-in's instance after its creation. It is there to allow a plugin to create any per-instance data structures it may need.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionDescribe* has been called
- the instance is fully constructed, with all objects requested in the describe actions (eg, parameters and clips) have been constructed and have had their initial values set. This means that if the values are being loaded from an old setup, that load should have taken place before the create instance action is called.

Post

- the instance pointer will be valid until the *kOfxActionDestroyInstance* action is passed to the plug-in with the same instance handle

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored, but all was well anyway
- *kOfxStatErrFatal*
- *kOfxStatErrMemory*, in which case this may be called again after a memory purge
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message if possible and the host should destroy the instance handle and not attempt to proceed further

kOfxActionDestroyInstance

This action is the last passed to a plug-in's instance before its destruction. It is there to allow a plugin to destroy any per-instance data structures it may have created.

- *kOfxStatOK*, the action was trapped and all was well,
- *kOfxStatReplyDefault*, the action was ignored as the effect had nothing to do,
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the handle,
- the instance has not had any of its members destroyed yet,

Post

- the instance pointer is no longer valid and any operation on it will be undefined

Returns

To some extent, what is returned is moot, a bit like throwing an exception in a C++ destructor, so the host should continue destruction of the instance regardless.

kOfxActionInstanceChanged

This action signals that something has changed in a plugin's instance, either by user action, the host or the plugin itself. All change actions are bracketed by a pair of *kOfxActionBeginInstanceChanged* and *kOfxActionEndInstanceChanged* actions. The **inArgs** property set is used to determine what was the thing inside the instance that was changed.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropType* The type of the thing that changed which will be one of..
 - *kOfxTypeParameter* Indicating a parameter's value has changed in some way
 - *kOfxTypeClip* A clip to an image effect has changed in some way (for Image Effect Plugins only)
 - *kOfxPropName* the name of the thing that was changed in the instance
 - *kOfxPropChangeReason* what triggered the change, which will be one of...
 - *kOfxChangeUserEdited* - the user or host changed the instance somehow and caused a change to something, this includes undo/redos, resets and loading values from files or presets,
 - *kOfxChangePluginEdited* - the plugin itself has changed the value of the instance in some action
 - *kOfxChangeTime* - the time has changed and this has affected the value of the object because it varies over time
 - *kOfxPropTime*
 - the effect time at which the change occurred (for Image Effect Plugins only)
 - *kOfxImageEffectPropRenderScale*
 - the render scale currently being applied to any image fetched from a clip (for Image Effect Plugins only)
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- *kOfxActionBeginInstanceChanged* has been called on the instance handle.

Post

- *kOfxActionEndInstanceChanged* will be called on the instance handle.

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionBeginInstanceChanged

The *kOfxActionBeginInstanceChanged* and *kOfxActionEndInstanceChanged* actions are used to bracket all *kOfxActionInstanceChanged* actions, whether a single change or multiple changes. Some changes to a plugin instance can be grouped logically (eg: a ‘reset all’ button resetting all the instance’s parameters), the begin/end instance changed actions allow a plugin to respond appropriately to a large set of changes. For example, a plugin that maintains a complex internal state can delay any changes to that state until all parameter changes have completed.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropChangeReason* what triggered the change, which will be one of...
 - *kOfxChangeUserEdited* - the user or host changed the instance somehow and caused a change to something, this includes undo/redos, resets and loading values from files or presets,
 - *kOfxChangePluginEdited* - the plugin itself has changed the value of the instance in some action
 - *kOfxChangeTime* - the time has changed and this has affected the value of the object because it varies over time
- **outArgs** – is redundant and is set to NULL

Post

- For *kOfxActionBeginInstanceChanged* , *kOfxActionCreateInstance* has been called on the instance handle.
- For *kOfxActionEndInstanceChanged* , *kOfxActionBeginInstanceChanged* has been called on the instance handle.
- *kOfxActionCreateInstance* has been called on the instance handle.

Post

- For *kOfxActionBeginInstanceChanged* , *kOfxActionInstanceChanged* will be called at least once on the instance handle.
- *kOfxActionEndInstanceChanged* will be called on the instance handle.

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionEndInstanceChanged

Action called after the end of a set of *kOfxActionEndInstanceChanged* actions, used with *kOfxActionBeginInstanceChanged* to bracket a grouped set of changes, see *kOfxActionBeginInstanceChanged*.

kOfxActionBeginInstanceEdit

This is called when an instance is *first* actively edited by a user, ie: and interface is open and parameter values and input clips can be modified. It is there so that effects can create private user interface structures when necessary. Note that some hosts can have multiple editors open on the same effect instance simultaneously.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Post

- *kOfxActionEndInstanceEdit* will be called when the last editor is closed on the instance

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionEndInstanceEdit

This is called when the *last* user interface on an instance closed. It is there so that effects can destroy private user interface structures when necessary. Note that some hosts can have multiple editors open on the same effect instance simultaneously, this will only be called when the last of those editors are closed.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionBeginInstanceEdit* has been called on the instance handle,

Post

- no user interface is open on the instance

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxPropAPIVersion

Property on the host descriptor, saying what API version of the API is being implemented.

- Type - int X N
- Property Set - host descriptor.

This is a version string that will specify which version of the API is being implemented by a host. It can have multiple values. For example “1.0”, “1.2.4” etc....

If this is not present, it is safe to assume that the version of the API is “1.0”.

kOfxPropTime

General property used to get/set the time of something.

- Type - double X 1
- Default - 0, if a settable property
- Property Set - commonly used as an argument to actions, input and output.

kOfxPropIsInteractive

Indicates if a host is actively editing the effect with some GUI.

- Type - int X 1
- Property Set - effect instance (read only)
- Valid Values - 0 or 1

If false the effect currently has no interface, however this may be because the effect is loaded in a background render host, or it may be loaded on an interactive host that has not yet opened an editor for the effect.

The output of an effect should only ever depend on the state of its parameters, not on the interactive flag. The interactive flag is more a courtesy flag to let a plugin know that it has an interace. If a plugin want's to have its behaviour dependant on the interactive flag, it can always make a secret parameter which shadows the state if the flag.

kOfxPluginPropFilePath

The file path to the plugin.

- Type - C string X 1
- Property Set - effect descriptor (read only)

This is a string that indicates the file path where the plug-in was found by the host. The path is in the native path format for the host OS (eg: UNIX directory separators are forward slashes, Windows ones are backslashes).

The path is to the bundle location, see `InstallationLocation`. eg: `‘/usr/OFX/Plugins/AcmePlugins/AcmeFantasticPlugin.ofx.bundle’`

kOfxPropInstanceData

A private data pointer that the plug-in can store its own data behind.

- Type - pointer X 1
- Property Set - plugin instance (read/write),
- Default - NULL

This data pointer is unique to each plug-in instance, so two instances of the same plug-in do not share the same data pointer. Use it to hang any needed private data structures.

kOfxPropType

General property, used to identify the kind of an object behind a handle.

- Type - ASCII C string X 1
- Property Set - any object handle (read only)
- Valid Values - currently this can be...
 - *kOfxTypeImageEffectHost*
 - *kOfxTypeImageEffect*
 - *kOfxTypeImageEffectInstance*
 - *kOfxTypeParameter*
 - *kOfxTypeParameterInstance*
 - *kOfxTypeClip*
 - *kOfxTypeImage*

kOfxPropName

Unique name of an object.

- Type - ASCII C string X 1
- Property Set - on many objects (descriptors and instances), see `PropertiesByObject` (read only)

This property is used to label objects uniquely among objects of that type. It is typically set when a plugin creates a new object with a function that takes a name.

kOfxPropVersion

Identifies a specific version of a host or plugin.

- Type - int X N
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - “0”
- Valid Values - positive integers

This is a multi dimensional integer property that represents the version of a host (host descriptor), or plugin (plugin descriptor). These represent a version number of the form ‘1.2.3.4’, with each dimension adding another ‘dot’ on the right.

A version is considered to be more recent than another if its ordered set of values is lexicographically greater than another, reading left to right. (ie: 1.2.4 is smaller than 1.2.6). Also, if the number of dimensions is different, then the values of the missing dimensions are considered to be zero (so 1.2.4 is greater than 1.2).

kOfxPropVersionLabel

Unique user readable version string of a plugin or host.

- Type - string X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - none, the host needs to set this
- Valid Values - ASCII string

This is purely for user feedback, a plugin or host should use *kOfxPropVersion* if they need to check for specific versions.

kOfxPropPluginDescription

Description of the plug-in to a user.

- Type - string X 1
- Property Set - plugin descriptor (read/write) and instance (read only)
- Default - “”
- Valid Values - UTF8 string

This is a string giving a potentially verbose description of the effect.

kOfxPropLabel

User visible name of an object.

- Type - UTF8 C string X 1
- Property Set - on many objects (descriptors and instances), see PropertiesByObject. Typically readable and writable in most cases.
- Default - the *kOfxPropName* the object was created with.

The label is what a user sees on any interface in place of the object’s name.

Note that resetting this will also reset *kOfxPropShortLabel* and *kOfxPropLongLabel*.

kOfxPropIcon

If set this tells the host to use an icon instead of a label for some object in the interface.

- Type - string X 2
- Property Set - various descriptors in the API
- Default - ""
- Valid Values - ASCII string

The value is a path is defined relative to the Resource folder that points to an SVG or PNG file containing the icon.

The first dimension, if set, will the name of and SVG file, the second a PNG file.

kOfxPropShortLabel

Short user visible name of an object.

- Type - UTF8 C string X 1
- Property Set - on many objects (descriptors and instances), see *PropertiesByObject*. Typically readable and writable in most cases.
- Default - initially *kOfxPropName*, but will be reset if *kOfxPropLabel* is changed.

This is a shorter version of the label, typically 13 character glyphs or less. Hosts should use this if they have limited display space for their object labels.

kOfxPropLongLabel

Long user visible name of an object.

- Type - UTF8 C string X 1
- Property Set - on many objects (descriptors and instances), see *PropertiesByObject*. Typically readable and writable in most cases.
- Default - initially *kOfxPropName*, but will be reset if *kOfxPropLabel* is changed.

This is a longer version of the label, typically 32 character glyphs or so. Hosts should use this if they have much display space for their object labels.

kOfxPropChangeReason

Indicates why a plug-in changed.

- Type - ASCII C string X 1
- Property Set - inArgs parameter on the *kOfxActionInstanceChanged* action.
- Valid Values - this can be...
 - *kOfxChangeUserEdited* - the user directly edited the instance somehow and caused a change to something, this includes undo/redos and resets
 - *kOfxChangePluginEdited* - the plug-in itself has changed the value of the object in some action

- *kOfxChangeTime* - the time has changed and this has affected the value of the object because it varies over time

Argument property for the *kOfxActionInstanceChanged* action.

kOfxPropEffectInstance

A pointer to an effect instance.

- Type - pointer X 1
- Property Set - on an interact instance (read only)

This property is used to link an object to the effect. For example if the plug-in supplies an OpenGL overlay for an image effect, the interact instance will have one of these so that the plug-in can connect back to the effect the GUI links to.

kOfxPropHostOSHandle

A pointer to an operating system specific application handle.

- Type - pointer X 1
- Property Set - host descriptor.

Some plug-in vendor want raw OS specific handles back from the host so they can do interesting things with host OS APIs. Typically this is to control windowing properly on Microsoft Windows. This property returns the appropriate 'root' window handle on the current operating system. So on Windows this would be the hWnd of the application main window.

kOfxChangeUserEdited

String used as a value to *kOfxPropChangeReason* to indicate a user has changed something.

kOfxChangePluginEdited

String used as a value to *kOfxPropChangeReason* to indicate the plug-in itself has changed something.

kOfxChangeTime

String used as a value to *kOfxPropChangeReason* to a time varying object has changed due to a time change.

kOfxFlagInfiniteMax

Used to flag infinite rects. Set minimums to this to indicate infinite.

This is effectively INT_MAX.

kOfxFlagInfiniteMin

Used to flag infinite rects. Set minimums to this to indicate infinite.

This is effectively INT_MIN

kOfxBitDepthNone

String used to label unset bitdepths.

kOfxBitDepthByte

String used to label unsigned 8 bit integer samples.

kOfxBitDepthShort

String used to label unsigned 16 bit integer samples.

kOfxBitDepthHalf

String used to label half-float (16 bit floating point) samples.

Version

Added in Version 1.4. Was in *ofxOpenGLRender.h* before.

kOfxBitDepthFloat

String used to label signed 32 bit floating point samples.

kOfxStatOK

Status code indicating all was fine.

kOfxStatFailed

Status error code for a failed operation.

kOfxStatErrFatal

Status error code for a fatal error.

Only returned in the case where the plug-in or host cannot continue to function and needs to be restarted.

kOfxStatErrUnknown

Status error code for an operation on or request for an unknown object.

kOfxStatErrMissingHostFeature

Status error code returned by plug-ins when they are missing host functionality, either an API or some optional functionality (eg: custom params).

Plug-Ins returning this should post an appropriate error message stating what they are missing.

kOfxStatErrUnsupported

Status error code for an unsupported feature/operation.

kOfxStatErrExists

Status error code for an operation attempting to create something that exists.

kOfxStatErrFormat

Status error code for an incorrect format.

kOfxStatErrMemory

Status error code indicating that something failed due to memory shortage.

kOfxStatErrBadHandle

Status error code for an operation on a bad handle.

kOfxStatErrBadIndex

Status error code indicating that a given index was invalid or unavailable.

kOfxStatErrValue

Status error code indicating that something failed due an illegal value.

kOfxStatReplyYes

OfxStatus returned indicating a 'yes'.

kOfxStatReplyNo

OfxStatus returned indicating a 'no'.

kOfxStatReplyDefault

OfxStatus returned indicating that a default action should be performed.

Typedefs

```
typedef struct OfxPropertySetStruct *OfxPropertySetHandle
```

Blind data structure to manipulate sets of properties through.

```
typedef int OfxStatus
```

OFX status return type.

```
typedef struct OfxHost OfxHost
```

Generic host structure passed to *OfxPlugin::setHost* function.

This structure contains what is needed by a plug-in to bootstrap its connection to the host.

```
OfxStatus() OfxPluginEntryPoint (const char *action, const void *handle,  
OfxPropertySetHandle inArgs, OfxPropertySetHandle outArgs)
```

Entry point for plug-ins.

- **action** ASCII c string indicating which action to take
- **instance** object to which action should be applied, this will need to be cast to the appropriate blind data type depending on the *action*
- **inData** handle that contains action specific properties
- **outData** handle where the plug-in should set various action specific properties

This is how the host generally communicates with a plug-in. Entry points are used to pass messages to various objects used within OFX. The main use is within the *OfxPlugin* struct.

The exact set of actions is determined by the plug-in API that is being implemented, however all plug-ins can perform several actions. For the list of actions consult OFX Actions.

typedef struct *OfxPlugin* **OfxPlugin**

The structure that defines a plug-in to a host.

This structure is the first element in any plug-in structure using the OFX plug-in architecture. By examining its members a host can determine the API that the plug-in implements, the version of that API, its name and version.

For details see Architecture.

typedef double **OfxTime**

How time is specified within the OFX API.

typedef struct *OfxRangeI* **OfxRangeI**

Defines one dimensional integer bounds.

typedef struct *OfxRangeD* **OfxRangeD**

Defines one dimensional double bounds.

typedef struct *OfxPointI* **OfxPointI**

Defines two dimensional integer point.

typedef struct *OfxPointD* **OfxPointD**

Defines two dimensional double point.

typedef struct *OfxRectI* **OfxRectI**

Defines two dimensional integer region.

Regions are $x1 \leq x < x2$

Infinite regions are flagged by setting

- $x1 = kOfxFlagInfiniteMin$
- $y1 = kOfxFlagInfiniteMin$
- $x2 = kOfxFlagInfiniteMax$
- $y2 = kOfxFlagInfiniteMax$

typedef struct *OfxRectD* **OfxRectD**

Defines two dimensional double region.

Regions are $x1 \leq x < x2$

Infinite regions are flagged by setting

- $x1 = kOfxFlagInfiniteMin$
- $y1 = kOfxFlagInfiniteMin$
- $x2 = kOfxFlagInfiniteMax$
- $y2 = kOfxFlagInfiniteMax$

Functions

OfxPlugin ***OfxGetPlugin**(int nth)

Returns the 'nth' plug-in implemented inside a binary.

Returns a pointer to the 'nth' plug-in implemented in the binary. A function of this type must be implemented in and exported from each plug-in binary.

int **OfxGetNumberOfPlugins**(void)

Defines the number of plug-ins implemented inside a binary.

A host calls this to determine how many plug-ins there are inside a binary it has loaded. A function of this type must be implemented in and exported from each plug-in binary.

OfxStatus **OfxSetHost**(const *OfxHost* *host)

First thing host should call.

This host call, added in 2020, is not specified in earlier implementation of the API. Therefore host must check if the plugin implemented it and not assume symbol exists. The order of calls is then: 1) *OfxSetHost*, 2) *OfxGetNumberOfPlugins*, 3) *OfxGetPlugin*. The host pointer is only assumed valid until *OfxGetPlugin* where it might get reset. Plug-in can return *kOfxStatFailed* to indicate it has nothing to do here, it's not for this Host and it should be skipped silently.

struct **OfxHost**

#include <ofxCore.h> Generic host structure passed to *OfxPlugin::setHost* function.

This structure contains what is needed by a plug-in to bootstrap its connection to the host.

Public Members

OfxPropertySetHandle **host**

Global handle to the host. Extract relevant host properties from this. This pointer will be valid while the binary containing the plug-in is loaded.

const void *(***fetchSuite**)(*OfxPropertySetHandle* host, const char *suiteName, int suiteVersion)

The function which the plug-in uses to fetch suites from the host.

- **host** the host the suite is being fetched from this *must* be the *host* member of the *OfxHost* struct containing *fetchSuite*.
- **suiteName** ASCII string labelling the host supplied API
- **suiteVersion** version of that suite to fetch

Any API fetched will be valid while the binary containing the plug-in is loaded.

Repeated calls to *fetchSuite* with the same parameters will return the same pointer.

It is recommended that hosts should return the same host and suite pointers to all plugins in the same shared lib or bundle.

returns

- NULL if the API is unknown (either the api or the version requested),
- pointer to the relevant API if it was found

struct **OfxPlugin**

#include <ofxCore.h> The structure that defines a plug-in to a host.

This structure is the first element in any plug-in structure using the OFX plug-in architecture. By examining its members a host can determine the API that the plug-in implements, the version of that API, its name and version.

For details see Architecture.

Public Members

const char ***pluginApi**

Defines the type of the plug-in, this will tell the host what the plug-in does. e.g.: an image effects plug-in would be a “OfxImageEffectPlugin”

int **apiVersion**

Defines the version of the pluginApi that this plug-in implements

const char ***pluginIdentifier**

String that uniquely labels the plug-in among all plug-ins that implement an API. It need not necessarily be human sensible, however the preference is to use reverse internet domain name of the developer, followed by a ‘.’ then by a name that represents the plug-in.. It must be a legal ASCII string and have no whitespace in the name and no non printing chars. For example “uk.co.somesoftwarehouse.myPlugin”

unsigned int **pluginVersionMajor**

Major version of this plug-in, this gets incremented when backwards compatibility is broken.

unsigned int **pluginVersionMinor**

Major version of this plug-in, this gets incremented when software is changed, but does not break backwards compatibility.

void (***setHost**)(OfxHost *host)

Function the host uses to connect the plug-in to the host’s api fetcher.

- **fetchApi** pointer to host’s API fetcher

Mandatory function.

The very first function called in a plug-in. The plug-in *must not* call any OFX functions within this, it must only set its local copy of the host pointer.

It is recommended that hosts should return the same host and suite pointers to all plugins in the same shared lib or bundle.

Pre

- nothing else has been called

Post

- the pointer suite is valid until the plug-in is unloaded

OfxPluginEntryPoint ***mainEntry**

Main entry point for plug-ins.

Mandatory function.

The exact set of actions is determined by the plug-in API that is being implemented, however all plug-ins can perform several actions. For the list of actions consult OFX Actions.

Preconditions

- setHost has been called

struct **OfxRangeI**

#include <ofxCore.h> Defines one dimensional integer bounds.

Public Members

int **min**

int **max**

struct **OfxRangeD**

#include <ofxCore.h> Defines one dimensional double bounds.

Public Members

double **min**

double **max**

struct **OfxPointI**

#include <ofxCore.h> Defines two dimensional integer point.

Public Members

int **x**

int **y**

struct **OfxPointD**

#include <ofxCore.h> Defines two dimensional double point.

Public Members

double **x**

double **y**

struct **OfxRectI**

#include <ofxCore.h> Defines two dimensional integer region.

Regions are $x_1 \leq x < x_2$

Infinite regions are flagged by setting

- $x_1 = kOfxFlagInfiniteMin$
- $y_1 = kOfxFlagInfiniteMin$
- $x_2 = kOfxFlagInfiniteMax$
- $y_2 = kOfxFlagInfiniteMax$

Public Members

int **x1**

int **y1**

int **x2**

int **y2**

struct **OfxRectD**

#include <ofxCore.h> Defines two dimensional double region.

Regions are $x_1 \leq x < x_2$

Infinite regions are flagged by setting

- $x_1 = kOfxFlagInfiniteMin$
- $y_1 = kOfxFlagInfiniteMin$
- $x_2 = kOfxFlagInfiniteMax$
- $y_2 = kOfxFlagInfiniteMax$

Public Members

double **x1**

double **y1**

double **x2**

double **y2**

File ofxDialog.h

This file contains an optional suite which should be used to popup a native OS dialog from a host parameter changed action.

When a host uses a fullscreen window and is running the OFX plugins in another thread it can lead to a lot of conflicts if that plugin will try to open its own window.

This suite will provide the functionality for a plugin to request running its dialog in the UI thread, and informing the host it will do this so it can take the appropriate actions needed. (Like lowering its priority etc..)

Defines

kOfxDialogSuite

The name of the Dialog suite, used to fetch from a host via *OfxHost::fetchSuite*.

kOfxActionDialog

Action called after a dialog has requested a 'Dialog' The arguments to the action are:

- **user_data** Pointer which was provided when the plugin requested the Dialog

When the plugin receives this action it is safe to popup a dialog. It runs in the host's UI thread, which may differ from the main OFX processing thread. Plugin should return from this action when all Dialog interactions are done. At that point the host will continue again. The host will not send any other messages asynchronous to this one.

Typedefs

```
typedef struct OfxDialogSuiteV1 OfxDialogSuiteV1
```

```
struct OfxDialogSuiteV1
```

```
    #include <ofxDialog.h>
```

Public Members

OfxStatus (***RequestDialog**)(void *user_data)

Request the host to send a `kOfxActionDialog` to the plugin from its UI thread.

Pre

- `user_data`: A pointer to any user data

Post

Return

- *kOfxStatOK* - The host has queued the request and will send an 'OfxActionDialog'
- *kOfxStatFailed* - The host has no provision for this or can not deal with it currently.

OfxStatus (***NotifyRedrawPending**)(void)

Inform the host of redraw event so it can redraw itself. If the host runs fullscreen in OpenGL, it would otherwise not receive redraw event when a dialog in front would catch all events.

Pre

Post

Return

- *kOfxStatReplyDefault*

File `ofxDrawSuite.h`

API for host- and GPU API-independent drawing.

Version

Added in OpenFX 1.5

Defines

`kOfxDrawSuite`

the string that names the DrawSuite, passed to *OfxHost::fetchSuite*

`kOfxInteractPropDrawContext`

The Draw Context handle.

- Type - pointer X 1
- Property Set - read only property on the inArgs of the following actions...
- *kOfxInteractActionDraw*

Typedefs

typedef struct OfxDrawContext ***OfxDrawContextHandle**
Blind declaration of an OFX drawing context.

typedef enum *OfxStandardColour* **OfxStandardColour**
Defines valid values for *OfxDrawSuiteV1::getColour*.

typedef enum *OfxDrawLineStipplePattern* **OfxDrawLineStipplePattern**
Defines valid values for *OfxDrawSuiteV1::setLineStipple*.

typedef enum *OfxDrawPrimitive* **OfxDrawPrimitive**
Defines valid values for *OfxDrawSuiteV1::draw*.

typedef enum *OfxDrawTextAlignment* **OfxDrawTextAlignment**
Defines text alignment values for *OfxDrawSuiteV1::drawText*.

typedef struct *OfxDrawSuiteV1* **OfxDrawSuiteV1**
OFX suite that allows an effect to draw to a host-defined display context.

Enums

enum **OfxStandardColour**
Defines valid values for *OfxDrawSuiteV1::getColour*.
Values:

enumerator **kOfxStandardColourOverlayBackground**

enumerator **kOfxStandardColourOverlayActive**

enumerator **kOfxStandardColourOverlaySelected**

enumerator **kOfxStandardColourOverlayDeselected**

enumerator **kOfxStandardColourOverlayMarqueeFG**

enumerator **kOfxStandardColourOverlayMarqueeBG**

enumerator **kOfxStandardColourOverlayText**

enum **OfxDrawLineStipplePattern**
Defines valid values for *OfxDrawSuiteV1::setLineStipple*.
Values:

enumerator **kOfxDrawLineStipplePatternSolid**

enumerator **kOfxDrawLineStipplePatternDot**

enumerator **kOfxDrawLineStipplePatternDash**

enumerator **kOfxDrawLineStipplePatternAltDash**

enumerator **kOfxDrawLineStipplePatternDotDash**

enum **OfxDrawPrimitive**

Defines valid values for *OfxDrawSuiteV1::draw*.

Values:

enumerator **kOfxDrawPrimitiveLines**

enumerator **kOfxDrawPrimitiveLineStrip**

enumerator **kOfxDrawPrimitiveLineLoop**

enumerator **kOfxDrawPrimitiveRectangle**

enumerator **kOfxDrawPrimitivePolygon**

enumerator **kOfxDrawPrimitiveEllipse**

enum **OfxDrawTextAlignment**

Defines text alignment values for *OfxDrawSuiteV1::drawText*.

Values:

enumerator **kOfxDrawTextAlignmentLeft**

enumerator **kOfxDrawTextAlignmentRight**

enumerator **kOfxDrawTextAlignmentTop**

enumerator **kOfxDrawTextAlignmentBottom**

enumerator **kOfxDrawTextAlignmentBaseline**

enumerator **kOfxDrawTextAlignmentCenterH**

enumerator **kOfxDrawTextAlignmentCenterV**

struct **OfxDrawSuiteV1**

#include <ofxDrawSuite.h> OFX suite that allows an effect to draw to a host-defined display context.

Public Members

OfxStatus (***getColor**)(*OfxDrawContextHandle* context, *OfxStandardColour* std_colour, *OfxRGBAColourF* *colour)

Retrieves the host's desired draw colour for.

- context draw context
- std_colour desired colour type
- colour returned RGBA colour

Return

- *kOfxStatOK* - the colour was returned
- *kOfxStatErrValue* - std_colour was invalid
- *kOfxStatFailed* - failure, e.g. if function is called outside kOfxInteractActionDraw

OfxStatus (***setColor**)(*OfxDrawContextHandle* context, const *OfxRGBAColourF* *colour)

Sets the colour for future drawing operations (lines, filled shapes and text)

- context draw context
- colour RGBA colour

The host should use “over” compositing when using a non-opaque colour.

Return

- *kOfxStatOK* - the colour was changed
- *kOfxStatFailed* - failure, e.g. if function is called outside kOfxInteractActionDraw

OfxStatus (***setLineWidth**)(*OfxDrawContextHandle* context, float width)

Sets the line width for future line drawing operations.

- context draw context
- width line width

Use width 0 for a single pixel line or non-zero for a smooth line of the desired width

The host should adjust for screen density.

Return

- *kOfxStatOK* - the width was changed

- *kOfxStatFailed* - failure, e.g. if function is called outside `kOfxInteractActionDraw`

OfxStatus (***setLineStipple**)(*OfxDrawContextHandle* context, *OfxDrawLineStipplePattern* pattern)

Sets the stipple pattern for future line drawing operations.

- `context` draw context
- `pattern` desired stipple pattern

Return

- *kOfxStatOK* - the pattern was changed
- *kOfxStatErrValue* - pattern was not valid
- *kOfxStatFailed* - failure, e.g. if function is called outside `kOfxInteractActionDraw`

OfxStatus (***draw**)(*OfxDrawContextHandle* context, *OfxDrawPrimitive* primitive, const *OfxPointD* *points, int point_count)

Draws a primitive of the desired type.

- `context` draw context
- `primitive` desired primitive
- `points` array of points in the primitive
- `point_count` number of points in the array

`kOfxDrawPrimitiveLines` - like `GL_LINES`, `n` points draws `n/2` separated lines
`kOfxDrawPrimitiveLineStrip` - like `GL_LINE_STRIP`, `n` points draws `n-1` connected lines
`kOfxDrawPrimitiveLineLoop` - like `GL_LINE_LOOP`, `n` points draws `n` connected lines
`kOfxDrawPrimitiveRectangle` - draws an axis-aligned filled rectangle defined by 2 opposite corner points
`kOfxDrawPrimitivePolygon` - like `GL_POLYGON`, draws a filled `n`-sided polygon
`kOfxDrawPrimitiveEllipse` - draws a axis-aligned elliptical line (not filled) within the rectangle defined by 2 opposite corner points

Return

- *kOfxStatOK* - the draw was completed
- *kOfxStatErrValue* - invalid primitive, or `point_count` not valid for primitive
- *kOfxStatFailed* - failure, e.g. if function is called outside `kOfxInteractActionDraw`

OfxStatus (***drawText**)(*OfxDrawContextHandle* context, const char *text, const *OfxPointD* *pos, int alignment)

Draws text at the specified position.

- `context` draw context
- `text` text to draw (UTF-8 encoded)
- `pos` position at which to align the text
- `alignment` text alignment flags (see `kOfxDrawTextAlignment*`)

The text font face and size are determined by the host.

Return

- *kOfxStatOK* - the text was drawn
- *kOfxStatErrValue* - text or pos were not defined
- *kOfxStatFailed* - failure, e.g. if function is called outside `kOfxInteractActionDraw`

File ofxGPURender.h

This file contains an optional suite **for** performing GPU-accelerated rendering of OpenFX Image Effect Plug-ins. For details see `\ref ofxGPURender`.

It allows hosts **and** plug-ins to support OpenGL, OpenCL, CUDA, **and** Metal. Additional GPU APIs, such as Vulkan, could use similar techniques.

StatusReturnValues

OfxStatus returns indicating that a OpenGL render error has occurred:

- If a plug-in returns *kOfxStatGLRenderFailed*, the host should retry the render with OpenGL rendering disabled.
- If a plug-in returns *kOfxStatGLOutOfMemory*, the host may choose to free resources on the GPU and retry the OpenGL render, rather than immediately falling back to CPU rendering.

kOfxStatGPUOutOfMemory

GPU render ran out of memory.

kOfxStatGLOutOfMemory

OpenGL render ran out of memory (same as `kOfxStatGPUOutOfMemory`)

kOfxStatGPURenderFailed

GPU render failed in a non-memory-related way.

kOfxStatGLRenderFailed

OpenGL render failed in a non-memory-related way (same as `kOfxStatGPURenderFailed`)

Defines

`__OFXGPURENDER_H__`

kOfxOpenGLRenderSuite

The name of the OpenGL render suite, used to fetch from a host via `OfxHost::fetchSuite`.

kOfxImageEffectPropOpenGLRenderSupported

Indicates whether a host or plug-in can support OpenGL accelerated rendering.

- Type - C string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only) - plug-in instance change (read/write)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - in which case the host or plug-in does not support OpenGL accelerated rendering
 - “true” - which means a host or plug-in can support OpenGL accelerated rendering, in the case of plug-ins this also means that it is capable of CPU based rendering in the absence of a GPU
 - “needed” - only for plug-ins, this means that an plug-in has to have OpenGL support, without which it cannot work.

V1.4: It is now expected from host reporting v1.4 that the plug-in can during instance change switch from true to false and false to true.

kOfxOpenGLPropPixelDepth

Indicates the bit depths supported by a plug-in during OpenGL renders.

This is analogous to *kOfxImageEffectPropSupportedPixelDepths*. When a plug-in sets this property, the host will try to provide buffers/textures in one of the supported formats. Additionally, the target buffers where the plug-in renders to will be set to one of the supported formats.

Unlike *kOfxImageEffectPropSupportedPixelDepths*, this property is optional. Shader-based effects might not really care about any format specifics when using OpenGL textures, so they can leave this unset and allow the host to decide the format.

- Type - string X N
- Property Set - plug-in descriptor (read only)
- Default - none set
- Valid Values - This must be one of
 - *kOfxBitDepthNone* (implying a clip is unconnected, not valid for an image)
 - *kOfxBitDepthByte*
 - *kOfxBitDepthShort*
 - *kOfxBitDepthHalf*
 - *kOfxBitDepthFloat*

kOfxImageEffectPropOpenGLEnabled

Indicates that a plug-in SHOULD use OpenGL acceleration in the current action.

When a plug-in and host have established they can both use OpenGL renders then when this property has been set the host expects the plug-in to render its result into the buffer it has setup before calling the render. The plug-in can then also safely use the ‘OfxImageEffectOpenGLRenderSuite’

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*

- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the plug-in cannot use the OpenGL suite
 - 1 indicates that the plug-in should render into the texture, and may use the OpenGL suite functions.

v1.4: *kOfxImageEffectPropOpenGLEnabled* should probably be checked in Instance Changed prior to try to read image via *clipLoadTexture*

Note: Once this property is set, the host and plug-in have agreed to use OpenGL, so the effect SHOULD access all its images through the OpenGL suite.

kOfxImageEffectPropOpenGLTextureIndex

Indicates the texture index of an image turned into an OpenGL texture by the host.

- Type - int X 1
- Property Set - texture handle returned by `OfxImageEffectOpenGLRenderSuiteV1::clipLoadTexture` (read only)

This value should be cast to a `GLuint` **and** used **as** the texture index when performing OpenGL texture operations.

The property set of the following actions should contain this property:

- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*

kOfxImageEffectPropOpenGLTextureTarget

Indicates the texture target enumerator of an image turned into an OpenGL texture by the host.

- Type - int X 1
- Property Set - texture handle returned by `OfxImageEffectOpenGLRenderSuiteV1::clipLoadTexture` (read only) This value should be cast to a `GLenum` and used as the texture target when performing OpenGL texture operations.

The property set of the following actions should contain this property:

- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*

kOfxActionOpenGLContextAttached

Action called when an effect has just been attached to an OpenGL context.

The purpose of this action is to allow a plug-in to set up any data it may need to do OpenGL rendering in an instance. For example...

- allocate a lookup table on a GPU,
- create an OpenCL or CUDA context that is bound to the host's OpenGL context so it can share buffers.

The plug-in will be responsible for deallocating any such shared resource in the kOfxActionOpenGLContextDetached action.

A host cannot call kOfxActionOpenGLContextAttached on the same instance without an intervening kOfxActionOpenGLContextDetached. A host can have a plug-in swap OpenGL contexts by issuing an attach/detach for the first context then another attach for the next context.

The arguments to the action are...

- **handle** handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** is redundant and set to NULL
- **outArgs** is redundant and set to NULL

A plug-in can return...

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored, but all was well anyway
- *kOfxStatErrMemory*, in which case this may be called again after a memory purge
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plug-in should post a message if possible and the host should not attempt to run the plug-in in OpenGL render mode.

kOfxActionOpenGLContextDetached

Action called when an effect is about to be detached from an OpenGL context.

The purpose of this action is to allow a plug-in to deallocate any resource allocated in kOfxActionOpenGLContextAttached just before the host decouples a plug-in from an OpenGL context. The host must call this with the same OpenGL context active as it called with the corresponding kOfxActionOpenGLContextAttached.

The arguments to the action are...

- **handle** handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** is redundant and set to NULL
- **outArgs** is redundant and set to NULL

A plug-in can return...

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored, but all was well anyway
- *kOfxStatErrMemory*, in which case this may be called again after a memory purge
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plug-in should post a message if possible and the host should not attempt to run the plug-in in OpenGL render mode.

kOfxImageEffectPropCudaRenderSupported

Indicates whether a host or plug-in can support CUDA render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support CUDA render
 - “true” - the host or plug-in can support CUDA render

kOfxImageEffectPropCudaEnabled

Indicates that a plug-in SHOULD use CUDA render in the current action.

If a plug-in and host have both set kOfxImageEffectPropCudaRenderSupported=“true” then the host MAY set this property to indicate that it is passing images as CUDA memory pointers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the kOfxImagePropData of each image of each clip is a CPU memory pointer.
 - 1 indicates that the kOfxImagePropData of each image of each clip is a CUDA memory pointer.

kOfxImageEffectPropCudaStreamSupported

Indicates whether a host or plug-in can support CUDA streams.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - in which case the host or plug-in does not support CUDA streams
 - “true” - which means a host or plug-in can support CUDA streams

kOfxImageEffectPropCudaStream

The CUDA stream to be used for rendering.

- Type - pointer X 1

- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property will only be set if the host and plug-in both support CUDA streams.

If set:

- this property contains a pointer to the stream of CUDA render (`cudaStream_t`). In order to use it, `reinterpret_cast<cudaStream_t>(pointer)` is needed.
- the plug-in SHOULD ensure that its render action enqueues any asynchronous CUDA operations onto the supplied queue.
- the plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action, and SHOULD NOT call `cudaDeviceSynchronize()` at any time.

If not set:

- the plug-in SHOULD ensure that any asynchronous operations it enqueues have completed before returning from the render action.

kOfxImageEffectPropMetalRenderSupported

Indicates whether a host or plug-in can support Metal render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support Metal render
 - “true” - the host or plug-in can support Metal render

kOfxImageEffectPropMetalEnabled

Indicates that a plug-in SHOULD use Metal render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropMetalRenderSupported=“true”` then the host MAY set this property to indicate that it is passing images as Metal buffers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values

- 0 indicates that the `kOfxImagePropData` of each image of each clip is a CPU memory pointer.
- 1 indicates that the `kOfxImagePropData` of each image of each clip is a Metal `id<MTLBuffer>`.

`kOfxImageEffectPropMetalCommandQueue`

The command queue of Metal render.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *`kOfxImageEffectActionRender`*
 - *`kOfxImageEffectActionBeginSequenceRender`*
 - *`kOfxImageEffectActionEndSequenceRender`*

This property contains a pointer to the command queue to be used for Metal rendering (`id<MTLCommandQueue>`). In order to use it, `reinterpret_cast<id<MTLCommandQueue>>(pointer)` is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous Metal operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

`kOfxImageEffectPropOpenCLRenderSupported`

Indicates whether a host or plug-in can support OpenCL Buffers render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - "false" for a plug-in
- Valid Values - This must be one of
 - "false" - the host or plug-in does not support OpenCL Buffers render
 - "true" - the host or plug-in can support OpenCL Buffers render

`kOfxImageEffectPropOpenCLSupported`

Indicates whether a host or plug-in can support OpenCL Images render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - "false" for a plug-in
- Valid Values - This must be one of
 - "false" - in which case the host or plug-in does not support OpenCL Images render
 - "true" - which means a host or plug-in can support OpenCL Images render

kOfxImageEffectPropOpenCLEnabled

Indicates that a plug-in SHOULD use OpenCL render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropOpenCLRenderSupported="true"` or have both set `kOfxImageEffectPropOpenCLSupported="true"` then the host MAY set this property to indicate that it is passing images as OpenCL Buffers or Images.

When rendering using OpenCL Buffers, the `cl_mem` of the buffers are retrieved using *kOfxImagePropData*. When rendering using OpenCL Images, the `cl_mem` of the images are retrieved using *kOfxImageEffectPropOpenCLImage*. If both *kOfxImageEffectPropOpenCLSupported* (Buffers) and *kOfxImageEffectPropOpenCLRenderSupported* (Images) are enabled by the plug-in, it should use *kOfxImageEffectPropOpenCLImage* to determine which is being used by the host.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that a plug-in SHOULD use OpenCL render in the render action
 - 1 indicates that a plug-in SHOULD NOT use OpenCL render in the render action

kOfxImageEffectPropOpenCLCommandQueue

Indicates the OpenCL command queue that should be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property contains a pointer to the command queue to be used for OpenCL rendering (`cl_command_queue`). In order to use it, `reinterpret_cast<cl_command_queue>(pointer)` is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous OpenCL operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

kOfxImageEffectPropOpenCLImage

Indicates the image handle of an image supplied as an OpenCL Image by the host.

- Type - pointer X 1
- Property Set - image handle returned by `clipGetImage`

This value should be cast to a `cl_mem` and used as the image handle when performing OpenCL Images operations. The property should be used (not `kOfxImagePropData`) when rendering with OpenCL Images (`kOfxImageEffectPropOpenCLSupported`), and should be used to determine whether Images or Buffers should be used if a plug-in supports both `kOfxImageEffectPropOpenCLSupported` and `kOfxImageEffectPropOpenCLRenderSupported`. Note: the `kOfxImagePropRowBytes` property is not required to be set by the host, since OpenCL Images do not have the concept of row bytes.

kOfxOpenCLProgramSuite

Typedefs

```
typedef struct OfxImageEffectOpenGLRenderSuiteV1 OfxImageEffectOpenGLRenderSuiteV1
```

OFX suite that provides image to texture conversion for OpenGL processing.

```
typedef struct OfxOpenCLProgramSuiteV1 OfxOpenCLProgramSuiteV1
```

OFX suite that allows a plug-in to get OpenCL programs compiled.

This is an optional suite the host can provide for building OpenCL programs for the plug-in, as an alternative to calling `clCreateProgramWithSource / clBuildProgram`. There are two advantages to doing this: The host can add flags (such as `-cl-denorms-are-zero`) to the build call, and may also cache program binaries for performance (however, if the source of the program or the OpenCL environment changes, the host must recompile so some mechanism such as hashing must be used).

```
struct OfxImageEffectOpenGLRenderSuiteV1
```

`#include <ofxGPURender.h>` OFX suite that provides image to texture conversion for OpenGL processing.

Public Members

```
OfxStatus (*clipLoadTexture)(OfxImageClipHandle clip, OfxTime time, const char *format, const OfxRectD *region, OfxPropertySetHandle *textureHandle)
```

loads an image from an OFX clip as a texture into OpenGL

- `clip` clip to load the image from
- `time` effect time to load the image from
- `format` requested texture format (As in none,byte,word,half,float, etc..) When set to NULL, the host decides the format based on the plug-in's `kOfxOpenGLPropPixelDepth` setting.
- `region` region of the image to load (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- `textureHandle` property set containing information about the texture

An image is fetched from a clip at the indicated time for the given region and loaded into an OpenGL texture. When a specific format is requested, the host ensures it gives the requested format. When the clip specified is the "Output" clip, the format is ignored and the host must bind the resulting texture as the current color buffer (render target). This may also be done prior to calling the `kOfxImageEffectActionRender` action. If the `region` parameter is set to non-NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set or is NULL, then the region fetched will be at least the Region of Interest the effect has previously specified,

clipped to the clip's Region of Definition. Information about the texture, including the texture index, is returned in the *textureHandle* argument. The properties on this handle will be...

- *kOfxImageEffectPropOpenGLTextureIndex*
- *kOfxImageEffectPropOpenGLTextureTarget*
- *kOfxImageEffectPropPixelDepth*
- *kOfxImageEffectPropComponents*
- *kOfxImageEffectPropPreMultiplication*
- *kOfxImageEffectPropRenderScale*
- *kOfxImagePropPixelAspectRatio*
- *kOfxImagePropBounds*
- *kOfxImagePropRegionOfDefinition*
- *kOfxImagePropRowBytes*
- *kOfxImagePropField*
- *kOfxImagePropUniqueIdentifier*

With the exception of the OpenGL specifics, these properties are the same as the properties in an image handle returned by *clipGetImage* in the image effect suite.

Note:

- this is the OpenGL equivalent of *clipGetImage* from *OfxImageEffectSuiteV1*
-

Pre

- clip was returned by *clipGetHandle*
- Format property in the texture handle

Post

- texture handle to be disposed of by *clipFreeTexture* before the action returns
- when the clip specified is the "Output" clip, the format is ignored and the host must bind the resulting texture as the current color buffer (render target). This may also be done prior to calling the render action.

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time and/or region, the plug-in should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - not enough OpenGL memory was available for the effect to load the texture. The plug-in should abort the GL render and return *kOfxStatErrMemory*, after which the host can decide to retry the operation with CPU based processing.

OfxStatus (***clipFreeTexture**)(*OfxPropertySetHandle* textureHandle)

Releases the texture handle previously returned by clipLoadTexture.

For input clips, this also deletes the texture from OpenGL. This should also be called on the output clip; for the Output clip, it just releases the handle but does not delete the texture (since the host will need to read it).

Pre

- textureHandle was returned by clipGetImage

Post

- all operations on textureHandle will be invalid, and the OpenGL texture it referred to has been deleted (for source clips)

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - general failure for some reason,
- *kOfxStatErrBadHandle* - the image handle was invalid,

OfxStatus (***flushResources**)()

Request the host to minimize its GPU resource load.

When a plug-in fails to allocate GPU resources, it can call this function to request the host to flush its GPU resources if it holds any. After the function the plug-in can try again to allocate resources which then might succeed if the host actually has released anything.

Pre

Post

- No changes to the plug-in GL state should have been made.

Return

- *kOfxStatOK* - the host has actually released some resources,
- *kOfxStatReplyDefault* - nothing the host could do..

struct **OfxOpenCLProgramSuiteV1**

#include <ofxGPURender.h> OFX suite that allows a plug-in to get OpenCL programs compiled.

This is an optional suite the host can provide for building OpenCL programs for the plug-in, as an alternative to calling `clCreateProgramWithSource` / `clBuildProgram`. There are two advantages to doing this: The host can add flags (such as `-cl-denorms-are-zero`) to the build call, and may also cache program binaries for performance (however, if the source of the program or the OpenCL environment changes, the host must recompile so some mechanism such as hashing must be used).

Public Members

OfxStatus (***compileProgram**)(const char *pszProgramSource, int fOptional, void *pResult)
Compiles the OpenCL program.

File ofxImageEffect.h

Defines

_ofxImageEffect_h_

kOfxImageEffectPluginApi

String used to label OFX Image Effect Plug-ins.

Set the pluginApi member of the OfxPluginHeader inside any OfxImageEffectPluginStruct to be this so that the host knows the plugin is an image effect.

kOfxImageEffectPluginApiVersion

The current version of the Image Effect API.

kOfxImageComponentNone

String to label something with unset components.

kOfxImageComponentRGBA

String to label images with RGBA components.

kOfxImageComponentRGB

String to label images with RGB components.

kOfxImageComponentAlpha

String to label images with only Alpha components.

kOfxImageEffectContextGenerator

Use to define the generator image effect context. See *kOfxImageEffectPropContext*.

kOfxImageEffectContextFilter

Use to define the filter effect image effect context See *kOfxImageEffectPropContext*.

kOfxImageEffectContextTransition

Use to define the transition image effect context See *kOfxImageEffectPropContext*.

kOfxImageEffectContextPaint

Use to define the paint image effect context See *kOfxImageEffectPropContext*.

kOfxImageEffectContextGeneral

Use to define the general image effect context See *kOfxImageEffectPropContext*.

kOfxImageEffectContextRetimer

Use to define the retimer effect context See *kOfxImageEffectPropContext*.

kOfxTypeImageEffectHost

Used as a value for *kOfxPropType* on image effect host handles.

kOfxTypeImageEffect

Used as a value for *kOfxPropType* on image effect plugin handles.

kOfxTypeImageEffectInstance

Used as a value for *kOfxPropType* on image effect instance handles

kOfxTypeClip

Used as a value for *kOfxPropType* on image effect clips.

kOfxTypeImage

Used as a value for *kOfxPropType* on image effect images.

kOfxImageEffectActionGetRegionOfDefinition

The region of definition for an image effect is the rectangular section of the 2D image plane that it is capable of filling, given the state of its input clips and parameters. This action is used to calculate the RoD for a plugin instance at a given frame. For more details on regions of definition see Image Effect Architectures.

Note that hosts that have constant sized imagery need not call this action, only hosts that allow image sizes to vary need call this.

If the effect did not trap this, it means the host should use the default RoD instead, which depends on the context. This is...

- generator context - defaults to the project window,
- filter and paint contexts - defaults to the RoD of the ‘Source’ input clip at the given time,
- transition context - defaults to the union of the RoDs of the ‘SourceFrom’ and ‘SourceTo’ input clips at the given time,
- general context - defaults to the union of the RoDs of all the non optional input clips and the ‘Source’ input clip (if it exists and it is connected) at the given time, if none exist, then it is the project window
- retimer context - defaults to the union of the RoD of the ‘Source’ input clip at the frame directly preceding the value of the ‘SourceTime’ double parameter and the frame directly after it

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the effect time for which a region of definition is being requested
 - *kOfxImageEffectPropRenderScale* the render scale that should be used in any calculations in this action

- **outArgs** – has the following property which the plug-in may set
 - *kOfxImageEffectPropRegionOfDefinition* the calculated region of definition, initially set by the host to the default RoD (see below), in Canonical Coordinates.

Returns

- *kOfxStatOK* the action was trapped and the RoD was set in the outArgs property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetRegionsOfInterest

This action allows a host to ask an effect, given a region I want to render, what region do you need from each of your input clips. In that way, depending on the host architecture, a host can fetch the minimal amount of the image needed as input. Note that there is a region of interest to be set in outArgs for each input clip that exists on the effect. For more details see Image EffectArchitectures”.

The default RoI is simply the value passed in on the *kOfxImageEffectPropRegionOfInterest* inArgs property set. All the RoIs in the outArgs property set must initialised to this value before the action is called.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the effect time for which a region of definition is being requested
 - *kOfxImageEffectPropRenderScale* the render scale that should be used in any calculations in this action
 - *kOfxImageEffectPropRegionOfInterest* the region to be rendered in the output image, in Canonical Coordinates.
- **outArgs** – has a set of 4 dimensional double properties, one for each of the input clips to the effect. The properties are each named *OfxImageClipPropRoI_* with the clip name post pended, for example *OfxImageClipPropRoI_Source*. These are initialised to the default RoI.

Returns

- *kOfxStatOK*, the action was trapped and at least one RoI was set in the outArgs property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetTimeDomain

This action allows a host to ask an effect what range of frames it can produce images over. Only effects instantiated in the GeneralContext” can have this called on them. In all other the host is in strict control over the temporal duration of the effect.

The default is:

- the union of all the frame ranges of the non optional input clips,
- infinite if there are no non optional input clips.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is null
- **outArgs** – has the following property
 - *kOfxImageEffectPropFrameRange* the frame range an effect can produce images for

Pre

- *kOfxActionCreateInstance* has been called on the instance
- the effect instance has been created in the general effect context

Returns

- *kOfxStatOK*, the action was trapped and the *kOfxImageEffectPropFrameRange* was set in the outArgs property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default value
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetFramesNeeded

This action lets the host ask the effect what frames are needed from each input clip to process a given frame. For example a temporal based degrainer may need several frames around the frame to render to do its work.

This action need only ever be called if the plugin has set the *kOfxImageEffectPropTemporalClipAccess* property on the plugin descriptor to be true. Otherwise the host assumes that the only frame needed from the inputs is the current one and this action is not called.

Note that each clip can have it's required frame range specified, and that you can specify discontinuous sets of ranges for each clip, for example

```
// The effect always needs the initial frame of the source as well as the previous_
↳and current frame
double rangeSource[4];

// required ranges on the source
rangeSource[0] = 0; // we always need frame 0 of the source
rangeSource[1] = 0;
rangeSource[2] = currentFrame - 1; // we also need the previous and current frame_
↳on the source
rangeSource[3] = currentFrame;

gPropHost->propSetDoubleN(outArgs, "OfxImageClipPropFrameRange_Source", 4,
↳rangeSource);
```

Which sets two discontinuous **range** of frames **from the 'Source'** clip required **as input**.

The default frame range is simply the single frame, `kOfxPropTime..kOfxPropTime`, found on the `inArgs` property set. All the frame ranges in the `outArgs` property set must be initialised to this value before the action is called.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following property
 - *kOfxPropTime* the effect time for which we need to calculate the frames needed on input
 - `outArgs` has a set of properties, one for each input clip, named `OfxImageClipPropFrameRange_` with the name of the clip post-pended. For example `OfxImageClipPropFrameRange_Source`. All these properties are multi-dimensional doubles, with the dimension is a multiple of two. Each pair of values indicates a continuous range of frames that is needed on the given input. They are all initialised to the default value.

Returns

- *kOfxStatOK*, the action was trapped and at least one frame range in the `outArgs` property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetClipPreferences

This action allows a plugin to dynamically specify its preferences for input and output clips. Please see Image Effect Clip Preferences for more details on the behaviour. Clip preferences are constant for the duration of an effect, so this action need only be called once per clip, not once per frame.

This should be called once after creation of an instance, each time an input clip is changed, and whenever a parameter named in the *kOfxImageEffectPropClipPreferencesSlaveParam* has its value changed.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – has the following properties which the plugin can set
 - a set of `char * X 1` properties, one for each of the input clips currently attached and the output clip, labelled with `OfxImageClipPropComponents_` post-pended with the clip's name. This must be set to one of the component types which the host supports and the effect stated it can accept on that input
 - a set of `char * X 1` properties, one for each of the input clips currently attached and the output clip, labelled with `OfxImageClipPropDepth_` post-pended with the clip's name. This must be set to one of the pixel depths both the host and plugin supports
 - a set of `double X 1` properties, one for each of the input clips currently attached and the output clip, labelled with `OfxImageClipPropPAR_` post-pended with the clip's name.

This is the pixel aspect ratio of the input and output clips. This must be set to a positive non zero double value,

- *kOfxImageEffectPropFrameRate* the frame rate of the output clip, this must be set to a positive non zero double value
- *kOfxImageClipPropFieldOrder* the fielding of the output clip
- *kOfxImageEffectPropPreMultiplication* the premultiplication of the output clip
- *kOfxImageClipPropContinuousSamples* whether the output clip can produce different images at non-frame intervals, defaults to false,
- *kOfxImageEffectFrameVarying* whether the output clip can produces different images at different times, even if all parameters and inputs are constant, defaults to false.

Returns

- *kOfxStatOK*, the action was trapped and at least one of the properties in the outArgs was changed from its default value
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionIsIdentity

Sometimes an effect can pass through an input uprocessed, for example a blur effect with a blur size of 0. This action can be called by a host before it attempts to render an effect to determine if it can simply copy input directly to output without having to call the render action on the effect.

If the effect does not need to process any pixels, it should set the value of the *kOfxPropName* to the clip that the host should us as the output instead, and the *kOfxPropTime* property on outArgs to be the time at which the frame should be fetched from a clip.

The default action is to call the render action on the effect.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the time at which to test for identity
 - *kOfxImageEffectPropFieldToRender* the field to test for identity
 - *kOfxImageEffectPropRenderWindow* the window (in \ref PixelCoordinates) to test for identity under
 - *kOfxImageEffectPropRenderScale* the scale factor being applied to the images being rendered
- **outArgs** – has the following properties which the plugin can set
 - *kOfxPropName* this to the name of the clip that should be used if the effect is an identity transform, defaults to the empty string
 - *kOfxPropTime* the time to use from the indicated source clip as an identity image (allowing time slips to happen), defaults to the value in *kOfxPropTime* in inArgs

Returns

- *kOfxStatOK*, the action was trapped and the effect should not have its render action called, the values in *outArgs* indicate what frame from which clip to use instead
- *kOfxStatReplyDefault*, the action was not trapped and the host should call the render action
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionRender

This action is where an effect gets to push pixels and turn its input clips and parameter set into an output image. This is possibly quite complicated and covered in the Rendering Image Effects chapter.

The render action *must* be trapped by the plug-in, it cannot return *kOfxStatReplyDefault*. The pixels needs be pushed I'm afraid.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the time at which to render
 - *kOfxImageEffectPropFieldToRender* the field to render
 - *kOfxImageEffectPropRenderWindow* the window (in \backslash ref PixelCoordinates) to render
 - *kOfxImageEffectPropRenderScale* the scale factor being applied to the images being rendered
 - *kOfxImageEffectPropSequentialRenderStatus* whether the effect is currently being rendered in strict frame order on a single instance
 - *kOfxImageEffectPropInteractiveRenderStatus* if the render is in response to a user modifying the effect in an interactive session
 - *kOfxImageEffectPropRenderQualityDraft* if the render should be done in draft mode (e.g. for faster scrubbing)
- **outArgs** – is redundant and should be set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance
- *kOfxImageEffectActionBeginSequenceRender* has been called on the instance

Post

- *kOfxImageEffectActionEndSequenceRender* action will be called on the instance

Returns

- *kOfxStatOK*, the effect rendered happily
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionBeginSequenceRender

This action is passed to an image effect before it renders a range of frames. It is there to allow an effect to set things up for a long sequence of frames. Note that this is still called, even if only a single frame is being rendered in an interactive environment.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxImageEffectPropFrameRange* the range of frames (inclusive) that will be rendered
 - *kOfxImageEffectPropFrameStep* what is the step between frames, generally set to 1 (for full frame renders) or 0.5 (for fielded renders)
 - *kOfxPropIsInteractive* is this a single frame render due to user interaction in a GUI, or a proper full sequence render.
 - *kOfxImageEffectPropRenderScale* the scale factor to apply to images for this call
 - *kOfxImageEffectPropSequentialRenderStatus* whether the effect is currently being rendered in strict frame order on a single instance
 - *kOfxImageEffectPropInteractiveRenderStatus* if the render is in response to a user modifying the effect in an interactive session
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance

Post

- *kOfxImageEffectActionRender* action will be called at least once on the instance
- *kOfxImageEffectActionEndSequenceRender* action will be called on the instance

Returns

- *kOfxStatOK*, the action was trapped and handled cleanly by the effect,
- *kOfxStatReplyDefault*, the action was not trapped, but all is well anyway,
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge,
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message,
- *kOfxStatErrFatal*

kOfxImageEffectActionEndSequenceRender

This action is passed to an image effect after it has rendered a range of frames. It is there to allow an effect to free resources after a long sequence of frame renders. Note that this is still called, even if only a single frame is being rendered in an interactive environment.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxImageEffectPropFrameRange* the range of frames (inclusive) that will be rendered

- *kOfxImageEffectPropFrameStep* what is the step between frames, generally set to 1 (for full frame renders) or 0.5 (for fielded renders),
- *kOfxPropIsInteractive*
- is this a single frame render due to user interaction in a GUI, or a proper full sequence render.
- *kOfxImageEffectPropRenderScale*
- the scale factor to apply to images for this call
- *kOfxImageEffectPropSequentialRenderStatus*
- whether the effect is currently being rendered in strict frame order on a single instance
- *kOfxImageEffectPropInteractiveRenderStatus*
- if the render is in response to a user modifying the effect in an interactive session
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance
- *kOfxImageEffectActionEndSequenceRender* action was called on the instance
- *kOfxImageEffectActionRender* action was called at least once on the instance

Returns

- *kOfxStatOK*, the action was trapped and handled cleanly by the effect,
- *kOfxStatReplyDefault*, the action was not trapped, but all is well anyway,
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge,
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message,
- *kOfxStatErrFatal*

kOfxImageEffectActionDescribeInContext

This action is unique to OFX Image Effect plug-ins. Because a plugin is able to exhibit different behaviour depending on the context of use, each separate context will need to be described individually. It is within this action that image effects describe which parameters and input clips it requires.

This action will be called multiple times, one for each of the contexts the plugin says it is capable of implementing. If a host does not support a certain context, then it need not call *kOfxImageEffectActionDescribeInContext* for that context.

This action *must* be trapped, it is not optional.

Parameters

- **handle** – handle to the context descriptor, cast to an *OfxImageEffectHandle* this may or may not be the same as passed to *kOfxActionDescribe*
- **inArgs** – has the following property:
 - *kOfxImageEffectPropContext* the context being described
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionDescribe* has been called on the descriptor handle,

- *kOfxActionCreateInstance* has not been called

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatErrMissingHostFeature*, in which the context will be ignored by the host, the plugin may post a message
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectPropSupportedContexts

Indicates to the host the contexts a plugin can be used in.

- Type - string X N
- Property Set - image effect descriptor passed to kOfxActionDescribe (read/write)
- Default - this has no defaults, it must be set
- Valid Values - This must be one of
 - *kOfxImageEffectContextGenerator*
 - *kOfxImageEffectContextFilter*
 - *kOfxImageEffectContextTransition*
 - *kOfxImageEffectContextPaint*
 - *kOfxImageEffectContextGeneral*
 - *kOfxImageEffectContextRetimer*

kOfxImageEffectPropPluginHandle

The plugin handle passed to the initial 'describe' action.

- Type - pointer X 1
- Property Set - plugin instance, (read only)

This value will be the same for all instances of a plugin.

kOfxImageEffectHostPropIsBackground

Indicates if a host is a background render.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 if the host is a foreground host, it may open the effect in an interactive session (or not)
 - 1 if the host is a background 'processing only' host, and the effect will never be opened in an interactive session.

kOfxImageEffectPluginPropSingleInstance

Indicates whether only one instance of a plugin can exist at the same time.

- Type - int X 1
- Property Set - plugin descriptor (read/write)
- Default - 0
- Valid Values - This must be one of
 - 0 - which means multiple instances can exist simultaneously,
 - 1 - which means only one instance can exist at any one time.

Some plugins, for whatever reason, may only be able to have a single instance in existence at any one time. This plugin property is used to indicate that.

kOfxImageEffectPluginRenderThreadSafety

Indicates how many simultaneous renders the plugin can deal with.

- Type - string X 1
- Property Set - plugin descriptor (read/write)
- Default - *kOfxImageEffectRenderInstanceSafe*
- Valid Values - This must be one of
 - *kOfxImageEffectRenderUnsafe* - indicating that only a single ‘render’ call can be made at any time among all instances,
 - *kOfxImageEffectRenderInstanceSafe* - indicating that any instance can have a single ‘render’ call at any one time,
 - *kOfxImageEffectRenderFullySafe* - indicating that any instance of a plugin can have multiple renders running simultaneously

kOfxImageEffectRenderUnsafe

String used to label render threads as un thread safe, see, *kOfxImageEffectPluginRenderThreadSafety*.

kOfxImageEffectRenderInstanceSafe

String used to label render threads as instance thread safe, *kOfxImageEffectPluginRenderThreadSafety*.

kOfxImageEffectRenderFullySafe

String used to label render threads as fully thread safe, *kOfxImageEffectPluginRenderThreadSafety*.

kOfxImageEffectPluginPropHostFrameThreading

Indicates whether a plugin lets the host perform per frame SMP threading.

- Type - int X 1
- Property Set - plugin descriptor (read/write)

- Default - 1
- Valid Values - This must be one of
 - 0 - which means that the plugin will perform any per frame SMP threading
 - 1 - which means the host can call an instance's render function simultaneously at the same frame, but with different windows to render.

kOfxImageEffectPropSupportsMultipleClipDepths

Indicates whether a host or plugin can support clips of differing component depths going into/out of an effect.

- Type - int X 1
- Property Set - plugin descriptor (read/write), host descriptor (read only)
- Default - 0 for a plugin
- Valid Values - This must be one of
 - 0 - in which case the host or plugin does not support clips of multiple pixel depths,
 - 1 - which means a host or plugin is able to deal with clips of multiple pixel depths,

If a host indicates that it can support multiple pixels depths, then it will allow the plugin to explicitly set the output clip's pixel depth in the *kOfxImageEffectActionGetClipPreferences* action. See *ImageEffectClipPreferences*.

kOfxImageEffectPropSupportsMultipleClipPARs

Indicates whether a host or plugin can support clips of differing pixel aspect ratios going into/out of an effect.

- Type - int X 1
- Property Set - plugin descriptor (read/write), host descriptor (read only)
- Default - 0 for a plugin
- Valid Values - This must be one of
 - 0 - in which case the host or plugin does not support clips of multiple pixel aspect ratios
 - 1 - which means a host or plugin is able to deal with clips of multiple pixel aspect ratios

If a host indicates that it can support multiple pixel aspect ratios, then it will allow the plugin to explicitly set the output clip's aspect ratio in the *kOfxImageEffectActionGetClipPreferences* action. See *ImageEffectClipPreferences*.

kOfxImageEffectPropClipPreferencesSlaveParam

Indicates the set of parameters on which a value change will trigger a change to clip preferences.

- Type - string X N
- Property Set - plugin descriptor (read/write)
- Default - none set
- Valid Values - the name of any described parameter

The plugin uses this to inform the host of the subset of parameters that affect the effect's clip preferences. A value change in any one of these will trigger a call to the clip preferences action.

The plugin can be slaved to multiple parameters (setting index 0, then index 1 etc...)

kOfxImageEffectPropSettableFrameRate

Indicates whether the host will let a plugin set the frame rate of the output clip.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 - in which case the plugin may not change the frame rate of the output clip,
 - 1 - which means a plugin is able to change the output clip's frame rate in the *kOfxImageEffectActionGetClipPreferences* action.

See ImageEffectClipPreferences.

If a clip can be continuously sampled, the frame rate will be set to 0.

kOfxImageEffectPropSettableFielding

Indicates whether the host will let a plugin set the fielding of the output clip.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 - in which case the plugin may not change the fielding of the output clip,
 - 1 - which means a plugin is able to change the output clip's fielding in the *kOfxImageEffectActionGetClipPreferences* action.

See ImageEffectClipPreferences.

kOfxImageEffectInstancePropSequentialRender

Indicates whether a plugin needs sequential rendering, and a host support it.

- Type - int X 1
- Property Set - plugin descriptor (read/write) or plugin instance (read/write), and host descriptor (read only)
- Default - 0
- Valid Values -
 - 0 - for a plugin, indicates that a plugin does not need to be sequentially rendered to be correct, for a host, indicates that it cannot ever guarantee sequential rendering,
 - 1 - for a plugin, indicates that it needs to be sequentially rendered to be correct, for a host, indicates that it can always support sequential rendering of plugins that are sequentially rendered,

- 2 - for a plugin, indicates that it is best to render sequentially, but will still produce correct results if not, for a host, indicates that it can sometimes render sequentially, and will have set *kOfxImageEffectPropSequentialRenderStatus* on the relevant actions

Some effects have temporal dependencies, some information from from the rendering of frame N-1 is needed to render frame N correctly. This property is set by an effect to indicate such a situation. Also, some effects are more efficient if they run sequentially, but can still render correct images even if they do not, eg: a complex particle system.

During an interactive session a host may attempt to render a frame out of sequence (for example when the user scrubs the current time), and the effect needs to deal with such a situation as best it can to provide feedback to the user.

However if a host caches output, any frame frame generated in random temporal order needs to be considered invalid and needs to be re-rendered when the host finally performs a first to last render of the output sequence.

In all cases, a host will set the *kOfxImageEffectPropSequentialRenderStatus* flag to indicate its sequential render status.

kOfxImageEffectPropSequentialRenderStatus

Property on all the render action that indicate the current sequential render status of a host.

- Type - int X 1
- Property Set - read only property on the inArgs of the following actions...
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values -
 - 0 - the host is not currently sequentially rendering,
 - 1 - the host is currently rendering in a way so that it guarantees sequential rendering.

This property is set to indicate whether the effect is currently being rendered in frame order on a single effect instance. See *kOfxImageEffectInstancePropSequentialRender* for more details on sequential rendering.

kOfxHostNativeOriginBottomLeft

kOfxHostNativeOriginTopLeft

kOfxHostNativeOriginCenter

kOfxImageEffectHostPropNativeOrigin

Property that indicates the host native UI space - this is only a UI hint, has no impact on pixel processing.

- Type - UTF8 string X 1
- Property Set - read only property (host)
 - Valid Values - “kOfxImageEffectHostPropNativeOriginBottomLeft” - 0,0 bottom left “kOfxImageEffectHostPropNativeOriginTopLeft” - 0,0 top left “kOfxImageEffectHostPropNativeOriginCenter” - 0,0 center (screen space)

This property is set to `kOfxHostNativeOriginBottomLeft` pre V1.4 and was to be discovered by plug-ins. This is useful for drawing overlay for points... so everything matches the rest of the app (for example expression linking to other tools, or simply match the reported location of the host viewer).

kOfxImageEffectPropInteractiveRenderStatus

Property that indicates if a plugin is being rendered in response to user interaction.

- Type - int X 1
- Property Set - read only property on the `inArgs` of the following actions...
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values -
 - 0 - the host is rendering the instance due to some reason other than an interactive tweak on a UI,
 - 1 - the instance is being rendered because a user is modifying parameters in an interactive session.

This property is set to 1 on all render calls that have been triggered because a user is actively modifying an effect (or up stream effect) in an interactive session. This typically means that the effect is not being rendered as a part of a sequence, but as a single frame.

kOfxImageEffectPluginPropGrouping

Indicates the effect group for this plugin.

- Type - UTF8 string X 1
- Property Set - plugin descriptor (read/write)
- Default - ""

This is purely a user interface hint for the host so it can group related effects on any menus it may have.

kOfxImageEffectPropSupportsOverlays

Indicates whether a host support image effect `ImageEffectOverlays`.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 - the host won't allow a plugin to draw a GUI over the output image,
 - 1 - the host will allow a plugin to draw a GUI over the output image.

kOfxImageEffectPluginPropOverlayInteractV1

Sets the entry for an effect's overlay interaction.

- Type - pointer X 1

- Property Set - plugin descriptor (read/write)
- Default - NULL
- Valid Values - must point to an *OfxPluginEntryPoint*

The entry point pointed to must be one that handles custom interaction actions.

kOfxImageEffectPluginPropOverlayInteractV2

Sets the entry for an effect's overlay interaction. Unlike `kOfxImageEffectPluginPropOverlayInteractV1`, the overlay interact in the plug-in is expected to implement the `kOfxInteractActionDraw` using the *OfxDrawSuiteV1*.

- Type - pointer X 1
- Property Set - plugin descriptor (read/write)
- Default - NULL
- Valid Values - must point to an *OfxPluginEntryPoint*

The entry point pointed to must be one that handles custom interaction actions.

kOfxImageEffectPropSupportsMultiResolution

Indicates whether a plugin or host support multiple resolution images.

- Type - int X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - 1 for plugins
- Valid Values - This must be one of
 - 0 - the plugin or host does not support multiple resolutions
 - 1 - the plugin or host does support multiple resolutions

Multiple resolution images mean...

- input and output images can be of any size
- input and output images can be offset from the origin

kOfxImageEffectPropSupportsTiles

Indicates whether a clip, plugin or host supports tiled images.

- Type - int X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write), clip descriptor (read/write), instance (read/write)
- Default - to 1 for a plugin and clip
- Valid Values - This must be one of 0 or 1

Tiled images mean that input or output images can contain pixel data that is only a subset of their full RoD.

If a clip or plugin does not support tiled images, then the host should supply full RoD images to the effect whenever it fetches one.

V1.4: It is now possible (defined) to change `OfxImageEffectPropSupportsTiles` in Instance Changed

kOfxImageEffectPropTemporalClipAccess

Indicates support for random temporal access to images in a clip.

- Type - int X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write), clip descriptor (read/write)
- Default - to 0 for a plugin and clip
- Valid Values - This must be one of 0 or 1

On a host, it indicates whether the host supports temporal access to images.

On a plugin, indicates if the plugin needs temporal access to images.

On a clip, it indicates that the clip needs temporal access to images.

kOfxImageEffectPropContext

Indicates the context a plugin instance has been created for.

- Type - string X 1
- Property Set - image effect instance (read only)
- Valid Values - This must be one of
 - *kOfxImageEffectContextGenerator*
 - *kOfxImageEffectContextFilter*
 - *kOfxImageEffectContextTransition*
 - *kOfxImageEffectContextPaint*
 - *kOfxImageEffectContextGeneral*
 - *kOfxImageEffectContextRetimer*

kOfxImageEffectPropPixelDepth

Indicates the type of each component in a clip or image (after any mapping)

- Type - string X 1
- Property Set - clip instance (read only), image instance (read only)
- Valid Values - This must be one of
 - `kOfxBitDepthNone` (implying a clip is unconnected, not valid for an image)
 - `kOfxBitDepthByte`
 - `kOfxBitDepthShort`
 - `kOfxBitDepthHalf`
 - `kOfxBitDepthFloat`

Note that for a clip, this is the value set by the clip preferences action, not the raw ‘actual’ value of the clip.

kOfxImageEffectPropComponents

Indicates the current component type in a clip or image (after any mapping)

- Type - string X 1
- Property Set - clip instance (read only), image instance (read only)
- Valid Values - This must be one of
 - kOfxImageComponentNone (implying a clip is unconnected, not valid for an image)
 - kOfxImageComponentRGBA
 - kOfxImageComponentRGB
 - kOfxImageComponentAlpha

Note that for a clip, this is the value set by the clip preferences action, not the raw ‘actual’ value of the clip.

kOfxImagePropUniqueIdentifier

Uniquely labels an image.

- Type - ASCII string X 1
- Property Set - image instance (read only)

This is host set and allows a plug-in to differentiate between images. This is especially useful if a plugin caches analysed information about the image (for example motion vectors). The plugin can label the cached information with this identifier. If a user connects a different clip to the analysed input, or the image has changed in some way then the plugin can detect this via an identifier change and re-evaluate the cached information.

kOfxImageClipPropContinuousSamples

Clip and action argument property which indicates that the clip can be sampled continuously.

- Type - int X 1
- Property Set - clip instance (read only), as an out argument to *kOfxImageEffectActionGetClipPreferences* action (read/write)
- Default - 0 as an out argument to the *kOfxImageEffectActionGetClipPreferences* action
- Valid Values - This must be one of...
 - 0 if the images can only be sampled at discreet times (eg: the clip is a sequence of frames),
 - 1 if the images can only be sampled continuously (eg: the clip is infact an animating roto spline and can be rendered anywhen).

If this is set to true, then the frame rate of a clip is effectively infinite, so to stop arithmetic errors the frame rate should then be set to 0.

kOfxImageClipPropUnmappedPixelDepth

Indicates the type of each component in a clip before any mapping by clip preferences.

- Type - string X 1

- Property Set - clip instance (read only)
- Valid Values - This must be one of
 - kOfxBitDepthNone (implying a clip is unconnected image)
 - kOfxBitDepthByte
 - kOfxBitDepthShort
 - kOfxBitDepthHalf
 - kOfxBitDepthFloat

This is the actual value of the component depth, before any mapping by clip preferences.

kOfxImageClipPropUnmappedComponents

Indicates the current 'raw' component type on a clip before any mapping by clip preferences.

- Type - string X 1
- Property Set - clip instance (read only),
- Valid Values - This must be one of
 - kOfxImageComponentNone (implying a clip is unconnected)
 - kOfxImageComponentRGBA
 - kOfxImageComponentRGB
 - kOfxImageComponentAlpha

kOfxImageEffectPropPreMultiplication

Indicates the premultiplication state of a clip or image.

- Type - string X 1
- Property Set - clip instance (read only), image instance (read only), out args property in the *kOfxImageEffectActionGetClipPreferences* action (read/write)
- Valid Values - This must be one of
 - kOfxImageOpaque - the image is opaque and so has no premultiplication state
 - kOfxImagePreMultiplied - the image is premultiplied by its alpha
 - kOfxImageUnPreMultiplied - the image is unpremultiplied

See the documentation on clip preferences for more details on how this is used with the *kOfxImageEffectActionGetClipPreferences* action.

kOfxImageOpaque

Used to flag the alpha of an image as opaque

kOfxImagePreMultiplied

Used to flag an image as premultiplied

kOfxImageUnPreMultiplied

Used to flag an image as unpremultiplied

kOfxImageEffectPropSupportedPixelDepths

Indicates the bit depths support by a plug-in or host.

- Type - string X N
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - plugin descriptor none set
- Valid Values - This must be one of
 - kOfxBitDepthNone (implying a clip is unconnected, not valid for an image)
 - kOfxBitDepthByte
 - kOfxBitDepthShort
 - kOfxBitDepthHalf
 - kOfxBitDepthFloat

The default for a plugin is to have none set, the plugin *must* define at least one in its describe action.

kOfxImageEffectPropSupportedComponents

Indicates the components supported by a clip or host,.

- Type - string X N
- Property Set - host descriptor (read only), clip descriptor (read/write)
- Valid Values - This must be one of
 - kOfxImageComponentNone (implying a clip is unconnected)
 - kOfxImageComponentRGBA
 - kOfxImageComponentRGB
 - kOfxImageComponentAlpha

This list of strings indicate what component types are supported by a host or are expected as input to a clip.

The default for a clip descriptor is to have none set, the plugin *must* define at least one in its define function

kOfxImageClipPropOptional

Indicates if a clip is optional.

- Type - int X 1
- Property Set - clip descriptor (read/write)
- Default - 0
- Valid Values - This must be one of 0 or 1

kOfxImageClipPropIsMask

Indicates that a clip is intended to be used as a mask input.

- Type - int X 1
- Property Set - clip descriptor (read/write)
- Default - 0
- Valid Values - This must be one of 0 or 1

Set this property on any clip which will only ever have single channel alpha images fetched from it. Typically on an optional clip such as a junk matte in a keyer.

This property acts as a hint to hosts indicating that they could feed the effect from a roto shape (or similar) rather than an 'ordinary' clip.

kOfxImagePropPixelAspectRatio

The pixel aspect ratio of a clip or image.

- Type - double X 1
- Property Set - clip instance (read only), image instance (read only) and *kOfxImageEffectActionGetClipPreferences* action out args property (read/write)

kOfxImageEffectPropFrameRate

The frame rate of a clip or instance's project.

- Type - double X 1
- Property Set - clip instance (read only), effect instance (read only) and *kOfxImageEffectActionGetClipPreferences* action out args property (read/write)

For an input clip this is the frame rate of the clip.

For an output clip, the frame rate mapped via pixel preferences.

For an instance, this is the frame rate of the project the effect is in.

For the outargs property in the *kOfxImageEffectActionGetClipPreferences* action, it is used to change the frame rate of the output clip.

kOfxImageEffectPropUnmappedFrameRate

Indicates the original unmapped frame rate (frames/second) of a clip.

- Type - double X 1
- Property Set - clip instance (read only),

If a plugin changes the output frame rate in the pixel preferences action, this property allows a plugin to get to the original value.

kOfxImageEffectPropFrameStep

The frame step used for a sequence of renders.

- Type - double X 1
- Property Set - an in argument for the *kOfxImageEffectActionBeginSequenceRender* action (read only)
- Valid Values - can be any positive value, but typically
 - 1 for frame based material
 - 0.5 for field based material

kOfxImageEffectPropFrameRange

The frame range over which a clip has images.

- Type - double X 2
- Property Set - clip instance (read only)

Dimension 0 is the first frame for which the clip can produce valid data.

Dimension 1 is the last frame for which the clip can produce valid data.

kOfxImageEffectPropUnmappedFrameRange

The unmaped frame range over which an output clip has images.

- Type - double X 2
- Property Set - clip instance (read only)

Dimension 0 is the first frame for which the clip can produce valid data.

Dimension 1 is the last frame for which the clip can produce valid data.

If a plugin changes the output frame rate in the pixel preferences action, it will affect the frame range of the output clip, this property allows a plugin to get to the original value.

kOfxImageClipPropConnected

Says whether the clip is actually connected at the moment.

- Type - int X 1
- Property Set - clip instance (read only)
- Valid Values - This must be one of 0 or 1

An instance may have a clip may not be connected to an object that can produce image data. Use this to find out.

Any clip that is not optional will *always* be connected during a render action. However, during interface actions, even non optional clips may be unconnected.

kOfxImageEffectFrameVarying

Indicates whether an effect will generate different images from frame to frame.

- Type - int X 1
- Property Set - out argument to *kOfxImageEffectActionGetClipPreferences* action (read/write).
- Default - 0
- Valid Values - This must be one of 0 or 1

This property indicates whether a plugin will generate a different image from frame to frame, even if no parameters or input image changes. For example a generator that creates random noise pixel at each frame.

kOfxImageEffectPropRenderScale

The proxy render scale currently being applied.

- Type - double X 2
- Property Set - an image instance (read only) and as read only an in argument on the following actions,
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
 - *kOfxImageEffectActionIsIdentity*
 - *kOfxImageEffectActionGetRegionOfDefinition*
 - *kOfxImageEffectActionGetRegionsOfInterest*
 - *kOfxActionInstanceChanged*
 - *kOfxInteractActionDraw*
 - *kOfxInteractActionPenMotion*
 - *kOfxInteractActionPenDown*
 - *kOfxInteractActionPenUp*
 - *kOfxInteractActionKeyDown*
 - *kOfxInteractActionKeyUp*
 - *kOfxInteractActionKeyRepeat*
 - *kOfxInteractActionGainFocus*
 - *kOfxInteractActionLoseFocus*

This should be applied to any spatial parameters to position them correctly. Not that the 'x' value does not include any pixel aspect ratios.

kOfxImageEffectPropRenderQualityDraft

Indicates whether an effect can take quality shortcuts to improve speed.

- Type - int X 1

- Property Set - render calls, host (read-only)
- Default - 0 - 0: Best Quality (1: Draft)
- Valid Values - This must be one of 0 or 1

This property indicates that the host provides the plug-in the option to render in Draft/Preview mode. This is useful for applications that must support fast scrubbing. These allow a plug-in to take short-cuts for improved performance when the situation allows and it makes sense, for example to generate thumbnails with effects applied. For example switch to a cheaper interpolation type or rendering mode. A plugin should expect frames rendered in this manner that will not be stucked in host cache unless the cache is only used in the same draft situations. If an host does not support that property a value of 0 is assumed. Also note that some hosts do implement `kOfxImageEffectPropRenderScale` - these two properties can be used independently.

kOfxImageEffectPropProjectExtent

The extent of the current project in canonical coordinates.

- Type - double X 2
- Property Set - a plugin instance (read only)

The extent is the size of the 'output' for the current project. See `NormalisedCoordinateSystem` for more information on the project extent.

The extent is in canonical coordinates and only returns the top right position, as the extent is always rooted at 0,0.

For example a PAL SD project would have an extent of 768, 576.

kOfxImageEffectPropProjectSize

The size of the current project in canonical coordinates.

- Type - double X 2
- Property Set - a plugin instance (read only)

The size of a project is a sub set of the *kOfxImageEffectPropProjectExtent*. For example a project may be a PAL SD project, but only be a letter-box within that. The project size is the size of this sub window.

The project size is in canonical coordinates.

See `NormalisedCoordinateSystem` for more information on the project extent.

kOfxImageEffectPropProjectOffset

The offset of the current project in canonical coordinates.

- Type - double X 2
- Property Set - a plugin instance (read only)

The offset is related to the *kOfxImageEffectPropProjectSize* and is the offset from the origin of the project 'sub-window'.

For example for a PAL SD project that is in letterbox form, the project offset is the offset to the bottom left hand corner of the letter box.

The project offset is in canonical coordinates.

See `NormalisedCoordinateSystem` for more information on the project extent.

`kOfxImageEffectPropProjectPixelAspectRatio`

The pixel aspect ratio of the current project.

- Type - double X 1
- Property Set - a plugin instance (read only)

`kOfxImageEffectInstancePropEffectDuration`

The duration of the effect.

- Type - double X 1
- Property Set - a plugin instance (read only)

This contains the duration of the plug-in effect, in frames.

`kOfxImageClipPropFieldOrder`

Which spatial field occurs temporally first in a frame.

- Type - string X 1
- Property Set - a clip instance (read only)
- Valid Values - This must be one of
 - *`kOfxImageFieldNone`* - the material is unfielded
 - *`kOfxImageFieldLower`* - the material is fielded, with image rows 0,2,4... occurring first in a frame
 - *`kOfxImageFieldUpper`* - the material is fielded, with image rows line 1,3,5... occurring first in a frame

`kOfxImagePropData`

The pixel data pointer of an image.

- Type - pointer X 1
- Property Set - an image instance (read only)

This property contains one of:

- a pointer to memory that is the lower left hand corner of an image
- a pointer to CUDA memory, if the Render action arguments includes `kOfxImageEffectPropCudaEnabled=1`
- an `id<MTLBuffer>`, if the Render action arguments includes `kOfxImageEffectPropMetalEnabled=1`
- a `cl_mem`, if the Render action arguments includes `kOfxImageEffectPropOpenCLEnabled=1`

See *`kOfxImageEffectPropCudaEnabled`*, *`kOfxImageEffectPropMetalEnabled`* and *`kOfxImageEffectPropOpenCLEnabled`*

kOfxImagePropBounds

The bounds of an image's pixels.

- Type - integer X 4
- Property Set - an image instance (read only)

The bounds, in PixelCoordinates, are of the addressable pixels in an image's data pointer.

The order of the values is x1, y1, x2, y2.

X values are $x1 \leq X < x2$ Y values are $y1 \leq Y < y2$

For less than full frame images, the pixel bounds will be contained by the *kOfxImagePropRegionOfDefinition* bounds.

kOfxImagePropRegionOfDefinition

The full region of definition of an image.

- Type - integer X 4
- Property Set - an image instance (read only)

An image's region of definition, in PixelCoordinates, is the full frame area of the image plane that the image covers.

The order of the values is x1, y1, x2, y2.

X values are $x1 \leq X < x2$ Y values are $y1 \leq Y < y2$

The *kOfxImagePropBounds* property contains the actual addressable pixels in an image, which may be less than its full region of definition.

kOfxImagePropRowBytes

The number of bytes in a row of an image.

- Type - integer X 1
- Property Set - an image instance (read only)

For various alignment reasons, a row of pixels may need to be padded at the end with several bytes before the next row starts in memory.

This property indicates the number of bytes in a row of pixels. This will be at least $\text{sizeof}(\text{PIXEL}) * (\text{bounds.x2} - \text{bounds.x1})$. Where bounds is fetched from the *kOfxImagePropBounds* property.

Note that (for CPU images only, not CUDA/Metal/OpenCL Buffers, nor OpenGL textures accessed via the OpenGL Render Suite) row bytes can be negative, which allows hosts with a native top down row order to pass image into OFX without having to repack pixels. Row bytes is not supported for OpenCL Images.

kOfxImagePropField

Which fields are present in the image.

- Type - string X 1

- Property Set - an image instance (read only)
- Valid Values - This must be one of
 - *kOfxImageFieldNone* - the image is an unfielded frame
 - *kOfxImageFieldBoth* - the image is fielded and contains both interlaced fields
 - *kOfxImageFieldLower* - the image is fielded and contains a single field, being the lower field (rows 0,2,4...)
 - *kOfxImageFieldUpper* - the image is fielded and contains a single field, being the upper field (rows 1,3,5...)

kOfxImageEffectPluginPropFieldRenderTwiceAlways

Controls how a plugin renders fielded footage.

- Type - integer X 1
- Property Set - a plugin descriptor (read/write)
- Default - 1
- Valid Values - This must be one of
 - 0 - the plugin is to have its render function called twice, only if there is animation in any of its parameters
 - 1 - the plugin is to have its render function called twice always

kOfxImageClipPropFieldExtraction

Controls how a plugin fetched fielded imagery from a clip.

- Type - string X 1
- Property Set - a clip descriptor (read/write)
- Default - *kOfxImageFieldDoubled*
- Valid Values - This must be one of
 - *kOfxImageFieldBoth* - fetch a full frame interlaced image
 - *kOfxImageFieldSingle* - fetch a single field, making a half height image
 - *kOfxImageFieldDoubled* - fetch a single field, but doubling each line and so making a full height image

This controls how a plug-in wishes to fetch images from a fielded clip, so it can tune its behaviour when it renders fielded footage.

Note that if it fetches *kOfxImageFieldSingle* and the host stores images natively as both fields interlaced, it can return a single image by doubling rowbytes and tweaking the starting address of the image data. This saves on a buffer copy.

kOfxImageEffectPropFieldToRender

Indicates which field is being rendered.

- Type - string X 1

- Property Set - a read only in argument property to *kOfxImageEffectActionRender* and *kOfxImageEffectActionIsIdentity*
- Valid Values - this must be one of
 - *kOfxImageFieldNone* - there are no fields to deal with, all images are full frame
 - *kOfxImageFieldBoth* - the imagery is fielded and both scan lines should be rendered
 - *kOfxImageFieldLower* - the lower field is being rendered (lines 0,2,4...)
 - *kOfxImageFieldUpper* - the upper field is being rendered (lines 1,3,5...)

kOfxImageEffectPropRegionOfDefinition

Used to indicate the region of definition of a plug-in.

- Type - double X 4
- Property Set - a read/write out argument property to the *kOfxImageEffectActionGetRegionOfDefinition* action
- Default - see *kOfxImageEffectActionGetRegionOfDefinition*

The order of the values is x1, y1, x2, y2.

This will be in CanonicalCoordinates

kOfxImageEffectPropRegionOfInterest

The value of a region of interest.

- Type - double X 4
- Property Set - a read only in argument property to the *kOfxImageEffectActionGetRegionsOfInterest* action

A host passes this value into the region of interest action to specify the region it is interested in rendering.

The order of the values is x1, y1, x2, y2.

This will be in CanonicalCoordinates.

kOfxImageEffectPropRenderWindow

The region to be rendered.

- Type - integer X 4
- Property Set - a read only in argument property to the *kOfxImageEffectActionRender* and *kOfxImageEffectActionIsIdentity* actions

The order of the values is x1, y1, x2, y2.

This will be in PixelCoordinates

kOfxImageFieldNone

String used to label imagery as having no fields

kOfxImageFieldLower

String used to label the lower field (scan lines 0,2,4...) of fielded imagery

kOfxImageFieldUpper

String used to label the upper field (scan lines 1,3,5...) of fielded imagery

kOfxImageFieldBoth

String used to label both fields of fielded imagery, indicating interlaced footage

kOfxImageFieldSingle

String used to label an image that consists of a single field, and so is half height

kOfxImageFieldDoubled

String used to label an image that consists of a single field, but each scan line is double, and so is full height

kOfxImageEffectOutputClipName

String that is the name of the standard OFX output clip.

kOfxImageEffectSimpleSourceClipName

String that is the name of the standard OFX single source input clip.

kOfxImageEffectTransitionSourceFromClipName

String that is the name of the 'from' clip in the OFX transition context.

kOfxImageEffectTransitionSourceToClipName

String that is the name of the 'from' clip in the OFX transition context.

kOfxImageEffectTransitionParamName

the name of the mandated 'Transition' param for the transition context

kOfxImageEffectRetimerParamName

the name of the mandated 'SourceTime' param for the retimer context

kOfxImageEffectSuite

the string that names image effect suites, passed to *OfxHost::fetchSuite*

kOfxStatErrImageFormat

Error code for incorrect image formats.

Typedefs

typedef struct OfxImageEffectStruct ***OfxImageEffectHandle**

Blind declaration of an OFX image effect.

typedef struct OfxImageClipStruct ***OfxImageClipHandle**

Blind declaration of an OFX image effect.

typedef struct OfxImageMemoryStruct ***OfxImageMemoryHandle**

Blind declaration for an handle to image memory returned by the image memory management routines.

typedef struct *OfxImageEffectSuiteV1* **OfxImageEffectSuiteV1**

The OFX suite for image effects.

This suite provides the functions needed by a plugin to defined and use an image effect plugin.

struct **OfxImageEffectSuiteV1**

#include <ofxImageEffect.h> The OFX suite for image effects.

This suite provides the functions needed by a plugin to defined and use an image effect plugin.

Public Members

OfxStatus (***getPropertySet**)(*OfxImageEffectHandle* imageEffect, *OfxPropertySetHandle* *propHandle)

Retrieves the property set for the given image effect.

- *imageEffect* image effect to get the property set for
- *propHandle* pointer to a the property set pointer, value is returned here

The property handle is for the duration of the image effect handle.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***getParamSet**)(*OfxImageEffectHandle* imageEffect, *OfiParamSetHandle* *paramSet)

Retrieves the parameter set for the given image effect.

- *imageEffect* image effect to get the property set for
- *paramSet* pointer to a the parameter set, value is returned here

The param set handle is valid for the lifetime of the image effect handle.

Return

- *kOfxStatOK* - the property set was found and returned

- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (*clipDefine)(*OfxImageEffectHandle* imageEffect, const char *name, *OfxPropertySetHandle* *propertySet)

Define a clip to the effect.

- *pluginHandle* handle passed into 'describeInContext' action
- *name* unique name of the clip to define
- *propertySet* property handle for the clip descriptor will be returned here

This function defines a clip to a host, the returned property set is used to describe various aspects of the clip to the host. Note that this does not create a clip instance.

Pre

- we are inside the describe in context action.

Return

OfxStatus (*clipGetHandle)(*OfxImageEffectHandle* imageEffect, const char *name, *OfxImageClipHandle* *clip, *OfxPropertySetHandle* *propertySet)

Get the property handle of the named input clip in the given instance.

- *imageEffect* an instance handle to the plugin
- *name* name of the clip, previously used in a clip define call
- *clip* where to return the clip
- *propertySet* if not NULL, the descriptor handle for a parameter's property set will be placed here.

The *propertySet* will have the same value as would be returned by *OfxImageEffectSuiteV1::clipGetPropertySet*

This **return** a clip handle **for** the given instance, note that this will **not** be the same **as** the clip handle returned by *clipDefine* **and** will be distant to clip handles **in any** other instance of the plugin.

Not a valid call **in any** of the describe actions.

Pre

- create instance action called,
- *name* passed to *clipDefine* for this context,
- not inside describe or describe in context actions.

Post

- handle will be valid for the life time of the instance.

OfxStatus (*clipGetPropertySet)(*OfxImageClipHandle* clip, *OfxPropertySetHandle* *propHandle)

Retrieves the property set for a given clip.

- clip clip effect to get the property set for
- propHandle pointer to a the property set handle, value is returned here

The property handle is valid for the lifetime of the clip, which is generally the lifetime of the instance.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (*clipGetImage)(*OfxImageClipHandle* clip, *OfxTime* time, const *OfxRectD* *region, *OfxPropertySetHandle* *imageHandle)

Get a handle for an image in a clip at the indicated time and indicated region.

- clip clip to extract the image from
- time time to fetch the image at
- region region to fetch the image from (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- imageHandle property set containing the image's data

An image is fetched from a clip at the indicated time for the given region and returned in the imageHandle.

If the *region* parameter is not set to NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped to the clip's Region of Definition.

If clipGetImage is called twice with the same parameters, then two separate image handles will be returned, each of which must be released. The underlying implementation could share image data pointers and use reference counting to maintain them.

Pre

- clip was returned by clipGetHandle

Post

- image handle is only valid for the duration of the action clipGetImage is called in
- image handle to be disposed of by clipReleaseImage before the action returns

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time and/or region, the plugin should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,

- *kOfxStatErrMemory* - the host had not enough memory to complete the operation, plugin should abort whatever it was doing.

OfxStatus (*clipReleaseImage)(*OfxPropertySetHandle* imageHandle)

Releases the image handle previously returned by clipGetImage.

Pre

- imageHandle was returned by clipGetImage

Post

- all operations on imageHandle will be invalid

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatErrBadHandle* - the image handle was invalid,

OfxStatus (*clipGetRegionOfDefinition)(*OfxImageClipHandle* clip, *OfxTime* time, *OfxRectD* *bounds)

Returns the spatial region of definition of the clip at the given time.

- clipHandle clip to extract the image from
- time time to fetch the image at
- region region to fetch the image from (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- imageHandle handle where the image is returned

An image is fetched from a clip at the indicated time for the given region and returned in the imageHandle.

If the *region* parameter is not set to NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped the clip's Region of Definition.

Pre

- clipHandle was returned by clipGetHandle

Post

- bounds will be filled the RoD of the clip at the indicated time

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time, the plugin should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - the host had not enough memory to complete the operation, plugin should abort whatever it was doing.

int (***abort**)(*OfxImageEffectHandle* imageEffect)

Returns whether to abort processing or not.

- *imageEffect* instance of the image effect

A host may want to signal to a plugin that it should stop whatever rendering it is doing and start again. Generally this is done in interactive threads in response to users tweaking some parameter.

This function indicates whether a plugin should stop whatever processing it is doing.

Return

- 0 if the effect should continue whatever processing it is doing
- 1 if the effect should abort whatever processing it is doing

OfxStatus (***imageMemoryAlloc**)(*OfxImageEffectHandle* instanceHandle, size_t nBytes, *OfxImageMemoryHandle* *memoryHandle)

Allocate memory from the host's image memory pool.

- *instanceHandle* effect instance to associate with this memory allocation, may be NULL.
- *nBytes* number of bytes to allocate
- *memoryHandle* pointer to the memory handle where a return value is placed

Memory handles allocated by this should be freed by *OfxImageEffectSuiteV1::imageMemoryFree*. To access the memory behind the handle you need to call *OfxImageEffectSuiteV1::imageMemoryLock*.

See ImageEffectsMemoryAllocation.

Return

- *kOfxStatOK* if all went well, a valid memory handle is placed in *memoryHandle*
- *kOfxStatErrBadHandle* if *instanceHandle* is not valid, *memoryHandle* is set to NULL
- *kOfxStatErrMemory* if there was not enough memory to satisfy the call, *memoryHandle* is set to NULL

OfxStatus (***imageMemoryFree**)(*OfxImageMemoryHandle* memoryHandle)

Frees a memory handle and associated memory.

- *memoryHandle* memory handle returned by *imageMemoryAlloc*

This function frees a memory handle and associated memory that was previously allocated via *OfxImageEffectSuiteV1::imageMemoryAlloc*

If there are outstanding locks, these are ignored and the handle and memory are freed anyway.

See ImageEffectsMemoryAllocation.

Return

- *kOfxStatOK* if the memory was cleanly deleted
- *kOfxStatErrBadHandle* if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*

OfxStatus (***imageMemoryLock**)(*OfxImageMemoryHandle* memoryHandle, void **returnedPtr)

Lock the memory associated with a memory handle and make it available for use.

- *memoryHandle* memory handle returned by *imageMemoryAlloc*
- *returnedPtr* where to the pointer to the locked memory

This function locks them memory associated with a memory handle and returns a pointer to it. The memory will be 16 byte aligned, to allow use of vector operations.

Note that memory locks and unlocks nest.

After the first lock call, the contents of the memory pointer to by *returnedPtr* is undefined. All subsequent calls to lock will return memory with the same contents as the previous call.

Also, if unlocked, then relocked, the memory associated with a memory handle may be at a different address.

See also *OfxImageEffectSuiteV1::imageMemoryUnlock* and *ImageEffectsMemoryAllocation*.

Return

- *kOfxStatOK* if the memory was locked, a pointer is placed in *returnedPtr*
- *kOfxStatErrBadHandle* if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*, null is placed in **returnedPtr*
- *kOfxStatErrMemory* if there was not enough memory to satisfy the call, **returnedPtr* is set to NULL

OfxStatus (***imageMemoryUnlock**)(*OfxImageMemoryHandle* memoryHandle)

Unlock allocated image data.

- *allocatedData* pointer to memory previously returned by *OfxImageEffectSuiteV1::imageAlloc*

This function unlocks a previously locked memory handle. Once completely unlocked, memory associated with a *memoryHandle* is no longer available for use. Attempting to use it results in undefined behaviour.

Note that locks and unlocks nest, and to fully unlock memory you need to match the count of locks placed upon it.

Also note, if you unlock a completely unlocked handle, it has no effect (ie: the lock count can't be negative).

If unlocked, then relocked, the memory associated with a memory handle may be at a different address, however the contents will remain the same.

See also *OfxImageEffectSuiteV1::imageMemoryLock* and *ImageEffectsMemoryAllocation*.

Return

- *kOfxStatOK* if the memory was unlocked cleanly,
- *kOfxStatErrBadHandle* if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*, null is placed in **returnedPtr*

File ofxInteract.h

Contains the API for ofx plugin defined GUIs and interaction.

Defines**kOfxInteractSuite****kOfxInteractPropSlaveToParam**

The set of parameters on which a value change will trigger a redraw for an interact.

- Type - string X N
- Property Set - interact instance property (read/write)
- Default - no values set
- Valid Values - the name of any parameter associated with this interact.

If the interact is representing the state of some set of OFX parameters, then it will need to be redrawn if any of those parameters' values change. This multi-dimensional property links such parameters to the interact.

The interact can be slaved to multiple parameters (setting index 0, then index 1 etc...)

kOfxInteractPropPixelScale

The size of a real screen pixel under the interact's canonical projection.

- Type - double X 2
- Property Set - interact instance and actions (read only)

kOfxInteractPropBackgroundColour

The background colour of the application behind an interact instance.

- Type - double X 3
- Property Set - read only on the interact instance and in argument to the *kOfxInteractActionDraw* action
- Valid Values - from 0 to 1

The components are in the order red, green then blue.

kOfxInteractPropSuggestedColour

The suggested colour to draw a widget in an interact, typically for overlays.

- Type - double X 3
- Property Set - read only on the interact instance
- Default - 1.0
- Valid Values - greater than or equal to 0.0

Some applications allow the user to specify colours of any overlay via a colour picker, this property represents the value of that colour. Plugins are at liberty to use this or not when they draw an overlay.

If a host does not support such a colour, it should return `kOfxStatReplyDefault`

kOfxInteractPropPenPosition

The position of the pen in an interact.

- Type - double X 2
- Property Set - read only in argument to the *kOfxInteractActionPenMotion*, *kOfxInteractActionPenDown* and *kOfxInteractActionPenUp* actions

This value passes the position of the pen into an interact. This is in the interact's canonical coordinates.

kOfxInteractPropPenViewportPosition

The position of the pen in an interact in viewport coordinates.

- Type - int X 2
- Property Set - read only in argument to the *kOfxInteractActionPenMotion*, *kOfxInteractActionPenDown* and *kOfxInteractActionPenUp* actions

This value passes the position of the pen into an interact. This is in the interact's OpenGL viewport coordinates, with 0,0 being at the bottom left.

kOfxInteractPropPenPressure

The pressure of the pen in an interact.

- Type - double X 1
- Property Set - read only in argument to the *kOfxInteractActionPenMotion*, *kOfxInteractActionPenDown* and *kOfxInteractActionPenUp* actions
- Valid Values - from 0 (no pressure) to 1 (maximum pressure)

This is used to indicate the status of the 'pen' in an interact. If a pen has only two states (eg: a mouse button), these should map to 0.0 and 1.0.

kOfxInteractPropBitDepth

Indicates whether the bits per component in the interact's OpenGL frame buffer.

- Type - int X 1
- Property Set - interact instance and descriptor (read only)

kOfxInteractPropHasAlpha

Indicates whether the interact's frame buffer has an alpha component or not.

- Type - int X 1

- Property Set - interact instance and descriptor (read only)
- Valid Values - This must be one of
 - 0 indicates no alpha component
 - 1 indicates an alpha component

kOfxActionDescribeInteract

This action is the first action passed to an interact. It is where an interact defines how it behaves and the resources it needs to function. If not trapped, the default action is for the host to carry on as normal. Note that the handle passed in acts as a descriptor for, rather than an instance of the interact.

Parameters

- **handle** – handle to the interact descriptor, cast to an *OfxInteractHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- The plugin has been loaded and the effect described.

Returns

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatErrMemory* in which case describe may be called again after a memory purge
- *kOfxStatFailed* something was wrong, the host should ignore the interact
- *kOfxStatErrFatal*

kOfxActionCreateInstanceInteract

This action is the first action passed to an interact instance after its creation. It is there to allow a plugin to create any per-instance data structures it may need.

Parameters

- **handle** – handle to the interact instance, cast to an *OfxInteractHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionDescribe* has been called on this interact

Post

- the instance pointer will be valid until the *kOfxActionDestroyInstance* action is passed to the plug-in with the same instance handle

Returns

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatReplyDefault* the action was ignored, but all was well anyway
- *kOfxStatErrFatal*
- *kOfxStatErrMemory* in which case this may be called again after a memory purge
- *kOfxStatFailed* in which case the host should ignore this interact

kOfxActionDestroyInstanceInteract

This action is the last passed to an interact's instance before its destruction. It is there to allow a plugin to destroy any per-instance data structures it may have created.

Parameters

- **handle** – handle to the interact instance, cast to an *OfxInteractHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the handle,
- the instance has not had any of its members destroyed yet

Post

- the instance pointer is no longer valid and any operation on it will be undefined

Returns

To some extent, what is returned is moot, a bit like throwing an exception in a C++ destructor, so the host should continue destruction of the instance regardless

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatReplyDefault* the action was ignored as the effect had nothing to do
- *kOfxStatErrFatal*
- *kOfxStatFailed* something went wrong, but no error code appropriate.

kOfxInteractActionDraw

This action is issued to an interact whenever the host needs the plugin to redraw the given interact.

The interact should either issue OpenGL calls to draw itself, or use DrawSuite calls.

If this is called via *kOfxImageEffectPluginPropOverlayInteractV2*, drawing MUST use DrawSuite.

If this is called via *kOfxImageEffectPluginPropOverlayInteractV1*, drawing SHOULD use OpenGL. Some existing plugins may use DrawSuite via *kOfxImageEffectPluginPropOverlayInteractV1* if it's supported by the host, but this is discouraged.

Note that the interact may (in the case of custom parameter GUIs) or may not (in the case of image effect overlays) be required to swap buffers, that is up to the kind of interact.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle
- the OpenGL context for this interact has been set
- the projection matrix will correspond to the interact's canonical view

Returns

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatReplyDefault* the action was ignored
- *kOfxStatErrFatal*
- *kOfxStatFailed* something went wrong, the host should ignore this interact in future

kOfxInteractActionPenMotion

This action is issued whenever the pen moves and the interact's has focus. It should be issued whether the pen is currently up or down. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
 - *kOfxInteractPropPenPosition* position of the pen in,
 - *kOfxInteractPropPenViewportPosition* position of the pen in,
 - *kOfxInteractPropPenPressure* the pressure of the pen,
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK* the host should not pass the pen motion to any other interactive object it may own that shares the same view.

Returns

- *kOfxStatOK* the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault* the action was not trapped and the host can deal with it if it wants

kOfxInteractActionPenDown

This action is issued when a pen transitions for the ‘up’ to the ‘down’ state. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin,
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
 - *kOfxInteractPropPenPosition* position of the pen in
 - *kOfxInteractPropPenViewportPosition* position of the pen in
 - *kOfxInteractPropPenPressure* the pressure of the pen
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same view.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault*, the action was not trapped and the host can deal with it if it wants

kOfxInteractActionPenUp

This action is issued when a pen transitions for the ‘down’ to the ‘up’ state. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin,
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred

- *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- *kOfxInteractPropPenPosition* position of the pen in
- *kOfxInteractPropPenViewportPosition* position of the pen in
- *kOfxInteractPropPenPressure* the pressure of the pen
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same view.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault*, the action was not trapped and the host can deal with it if it wants

kOfxInteractActionKeyDown

This action is issued when a key on the keyboard is depressed. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxPropKeySym* single integer value representing the key that was manipulated, this may not have a UTF8 representation (eg: a return key)
 - *kOfxPropKeyString* UTF8 string representing a character key that was pressed, some keys have no UTF8 encoding, in which case this is ""
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same focus.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own

- *kOfxStatReplyDefault* , the action was not trapped and the host can deal with it if it wants

kOfxInteractActionKeyUp

This action is issued when a key on the keyboard is released. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxPropKeySym* single integer value representing the key that was manipulated, this may not have a UTF8 representation (eg: a return key)
 - *kOfxPropKeyString* UTF8 string representing a character key that was pressed, some keys have no UTF8 encoding, in which case this is ""
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same focus.

Returns

- *kOfxStatOK* , the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault* , the action was not trapped and the host can deal with it if it wants

kOfxInteractActionKeyRepeat

This action is issued when a key on the keyboard is repeated. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxPropKeySym* single integer value representing the key that was manipulated, this may not have a UTF8 representation (eg: a return key)
 - *kOfxPropKeyString* UTF8 string representing a character key that was pressed, some keys have no UTF8 encoding, in which case this is ""
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched

- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same focus.

Returns

- *kOfxStatOK* , the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault* , the action was not trapped and the host can deal with it if it wants

kOfxInteractActionGainFocus

This action is issued when an interact gains input focus. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact is being used on,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels,
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Returns

- *kOfxStatOK* the action was trapped
- *kOfxStatReplyDefault* the action was not trapped

kOfxInteractActionLoseFocus

This action is issued when an interact loses input focus. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact is being used on,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels,

- *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
- *kOfxPropTime* the effect time at which changed occurred
- *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Returns

- *kOfxStatOK* the action was trapped
- *kOfxStatReplyDefault* the action was not trapped

Typedefs

typedef struct OfxInteract ***OfxInteractHandle**

Blind declaration of an OFX interactive gui.

typedef struct *OfxInteractSuiteV1* **OfxInteractSuiteV1**

OFX suite that allows an effect to interact with an openGL window so as to provide custom interfaces.

struct **OfxInteractSuiteV1**

#include <ofxInteract.h> OFX suite that allows an effect to interact with an openGL window so as to provide custom interfaces.

Public Members

OfxStatus (***interactSwapBuffers**)(*OfxInteractHandle* interactInstance)

Requests an openGL buffer swap on the interact instance.

OfxStatus (***interactRedraw**)(*OfxInteractHandle* interactInstance)

Requests a redraw of the interact instance.

OfxStatus (***interactGetPropertySet**)(*OfxInteractHandle* interactInstance, *OfxPropertySetHandle* *property)

Gets the property set handle for this interact handle.

File ofxKeySyms.h

Defines

kOfxPropKeySym

Property used to indicate which a key on the keyboard or a button on a button device has been pressed.

- Type - int X 1
- Property Set - an read only in argument for the actions *kOfxInteractActionKeyDown*, *kOfxInteractActionKeyUp* and *kOfxInteractActionKeyRepeat*.
- Valid Values - one of any specified by #defines in the file *ofxKeySyms.h*.

This property represents a raw key press, it does not represent the ‘character value’ of the key.

This property is associated with a *kOfxPropKeyString* property, which encodes the UTF8 value for the key-press/button press. Some keys (for example arrow keys) have no UTF8 equivalent.

Some keys, especially on non-english language systems, may have a UTF8 value, but *not* a keysym values, in these cases, the keysym will have a value of *kOfxKey_Unknown*, but the *kOfxPropKeyString* property will still be set with the UTF8 value.

kOfxPropKeyString

This property encodes a single keypresses that generates a unicode code point. The value is stored as a UTF8 string.

- Type - C string X 1, UTF8
- Property Set - an read only in argument for the actions *kOfxInteractActionKeyDown*, *kOfxInteractActionKeyUp* and *kOfxInteractActionKeyRepeat*.
- Valid Values - a UTF8 string representing a single character, or the empty string.

This property represents the UTF8 encode value of a single key press by a user in an OFX interact.

This property is associated with a *kOfxPropKeySym* which represents an integer value for the key press. Some keys (for example arrow keys) have no UTF8 equivalent, in which case this is set to the empty string “”, and the associate *kOfxPropKeySym* is set to the equivalent raw key press.

Some keys, especially on non-english language systems, may have a UTF8 value, but *not* a keysym values, in these cases, the keysym will have a value of *kOfxKey_Unknown*, but the *kOfxPropKeyString* property will still be set with the UTF8 value.

kOfxKey_Unknown

kOfxKey_BackSpace

kOfxKey_Tab

kOfxKey_Linefeed

kOfxKey_Clear

kOfxKey_Return

kOfxKey_Pause

kOfxKey_Scroll_Lock

kOfxKey_Sys_Req

kOfxKey_Escape

kOfxKey_Delete

kOfxKey_Multi_key

kOfxKey_SingleCandidate

kOfxKey_MultipleCandidate

kOfxKey_PreviousCandidate

kOfxKey_Kanji

kOfxKey_Muhenkan

kOfxKey_Henkan_Mode

kOfxKey_Henkan

kOfxKey_Romaji

kOfxKey_Hiragana

kOfxKey_Katakana

kOfxKey_Hiragana_Katakana

kOfxKey_Zenkaku

kOfxKey_Hankaku

kOfxKey_Zenkaku_Hankaku

kOfxKey_Touroku

kOfxKey_Massyo

kOfxKey_Kana_Lock

kOfxKey_Kana_Shift

kOfxKey_Eisu_Shift

kOfxKey_Eisu_toggle

kOfxKey_Zen_Koho

kOfxKey_Mae_Koho

kOfxKey_Home

kOfxKey_Left

kOfxKey_Up

kOfxKey_Right

kOfxKey_Down

kOfxKey_Prior

kOfxKey_Page_Up

kOfxKey_Next

kOfxKey_Page_Down

kOfxKey_End

kOfxKey_Begin

kOfxKey_Select

kOfxKey_Print

kOfxKey_Execute

kOfxKey_Insert

kOfxKey_Undo

kOfxKey_Redo

kOfxKey_Menu

kOfxKey_Find

kOfxKey_Cancel

kOfxKey_Help

kOfxKey_Break

kOfxKey_Mode_switch

kOfxKey_script_switch

kOfxKey_Num_Lock

kOfxKey_KP_Space

kOfxKey_KP_Tab

kOfxKey_KP_Enter

kOfxKey_KP_F1

kOfxKey_KP_F2

kOfxKey_KP_F3

kOfxKey_KP_F4

kOfxKey_KP_Home

kOfxKey_KP_Left

kOfxKey_KP_Up

kOfxKey_KP_Right

kOfxKey_KP_Down

kOfxKey_KP_Prior

kOfxKey_KP_Page_Up

kOfxKey_KP_Next

kOfxKey_KP_Page_Down

kOfxKey_KP_End

kOfxKey_KP_Begin

kOfxKey_KP_Insert

kOfxKey_KP_Delete

kOfxKey_KP_Equal

kOfxKey_KP_Multiply

kOfxKey_KP_Add

kOfxKey_KP_Separator

kOfxKey_KP_Subtract

kOfxKey_KP_Decimal

kOfxKey_KP_Divide

kOfxKey_KP_0

kOfxKey_KP_1

kOfxKey_KP_2

kOfxKey_KP_3

kOfxKey_KP_4

kOfxKey_KP_5

kOfxKey_KP_6

kOfxKey_KP_7

kOfxKey_KP_8

kOfxKey_KP_9

kOfxKey_F1

kOfxKey_F2

kOfxKey_F3

kOfxKey_F4

kOfxKey_F5

kOfxKey_F6

kOfxKey_F7

kOfxKey_F8

kOfxKey_F9

kOfxKey_F10

kOfxKey_F11

kOfxKey_L1

kOfxKey_F12

kOfxKey_L2

kOfxKey_F13

kOfxKey_L3

kOfxKey_F14

kOfxKey_L4

kOfxKey_F15

kOfxKey_L5

kOfxKey_F16

kOfxKey_L6

kOfxKey_F17

kOfxKey_L7

kOfxKey_F18

kOfxKey_L8

kOfxKey_F19

kOfxKey_L9

kOfxKey_F20

kOfxKey_L10

kOfxKey_F21

kOfxKey_R1

kOfxKey_F22

kOfxKey_R2

kOfxKey_F23

kOfxKey_R3

kOfxKey_F24

kOfxKey_R4

kOfxKey_F25

kOfxKey_R5

kOfxKey_F26

kOfxKey_R6

kOfxKey_F27

kOfxKey_R7

kOfxKey_F28

kOfxKey_R8

kOfxKey_F29

kOfxKey_R9

kOfxKey_F30

kOfxKey_R10

kOfxKey_F31

kOfxKey_R11

kOfxKey_F32

kOfxKey_R12

kOfxKey_F33

kOfxKey_R13

kOfxKey_F34

kOfxKey_R14

kOfxKey_F35

kOfxKey_R15

kOfxKey_Shift_L

kOfxKey_Shift_R

kOfxKey_Control_L

kOfxKey_Control_R

kOfxKey_Caps_Lock

kOfxKey_Shift_Lock

kOfxKey_Meta_L

kOfxKey_Meta_R

kOfxKey_Alt_L

kOfxKey_Alt_R

kOfxKey_Super_L

kOfxKey_Super_R

kOfxKey_Hyper_L

kOfxKey_Hyper_R

kOfxKey_space

kOfxKey_exclam

kOfxKey_quotedbl

kOfxKey_numbersign

kOfxKey_dollar

kOfxKey_percent

kOfxKey_ampersand

kOfxKey_apostrophe

kOfxKey_quoteright

kOfxKey_parenleft

kOfxKey_parenright

kOfxKey_asterisk

kOfxKey_plus

kOfxKey_comma

kOfxKey_minus

kOfxKey_period

kOfxKey_slash

kOfxKey_0

kOfxKey_1

kOfxKey_2

kOfxKey_3

kOfxKey_4

kOfxKey_5

kOfxKey_6

kOfxKey_7

kOfxKey_8

kOfxKey_9

kOfxKey_colon

kOfxKey_semicolon

kOfxKey_less

kOfxKey_equal

kOfxKey_greater

kOfxKey_question

kOfxKey_at

kOfxKey_A

kOfxKey_B

kOfxKey_C

kOfxKey_D

kOfxKey_E

kOfxKey_F

kOfxKey_G

kOfxKey_H

kOfxKey_I

kOfxKey_J

kOfxKey_K

kOfxKey_L

kOfxKey_M

kOfxKey_N

kOfxKey_O

kOfxKey_P

kOfxKey_Q

kOfxKey_R

kOfxKey_S

kOfxKey_T

kOfxKey_U

kOfxKey_V

kOfxKey_W

kOfxKey_X

kOfxKey_Y

kOfxKey_Z

kOfxKey_bracketleft

kOfxKey_backslash

kOfxKey_bracketright

kOfxKey_asciicircum

kOfxKey_underscore

kOfxKey_grave

kOfxKey_quoteleft

kOfxKey_a

kOfxKey_b

kOfxKey_c

kOfxKey_d

kOfxKey_e

kOfxKey_f

kOfxKey_g

kOfxKey_h

kOfxKey_i

kOfxKey_j

kOfxKey_k

kOfxKey_l

kOfxKey_m

kOfxKey_n

kOfxKey_o

kOfxKey_p

kOfxKey_q

kOfxKey_r

kOfxKey_s

kOfxKey_t

kOfxKey_u

kOfxKey_v

kOfxKey_w

kOfxKey_x

kOfxKey_y

kOfxKey_z

kOfxKey_braceleft

kOfxKey_bar

kOfxKey_braceright

kOfxKey_asciitilde

kOfxKey_nobreakspace

kOfxKey_exclamdown

kOfxKey_cent

kOfxKey_sterling

kOfxKey_currency

kOfxKey_yen

kOfxKey_brokenbar

kOfxKey_section

kOfxKey_diaeresis

kOfxKey_copyright

kOfxKey_ordfeminine

kOfxKey_guillemotleft

kOfxKey_notsign

kOfxKey_hyphen

kOfxKey_registered

kOfxKey_macron

kOfxKey_degree

kOfxKey_plusminus

kOfxKey_twosuperior

kOfxKey_threesuperior

kOfxKey_acute

kOfxKey_mu

kOfxKey_paragraph

kOfxKey_periodcentered

kOfxKey_cedilla

kOfxKey_onesuperior

kOfxKey_masculine

kOfxKey_guillemotright

kOfxKey_onequarter

kOfxKey_onehalf

kOfxKey_threequarters

kOfxKey_questiondown

kOfxKey_Agrave

kOfxKey_Aacute

kOfxKey_Acircumflex

kOfxKey_Atilde

kOfxKey_Adiaeresis

kOfxKey_Aring

kOfxKey_AE

kOfxKey_Ccedilla

kOfxKey_Egrave

kOfxKey_Eacute

kOfxKey_Ecircumflex

kOfxKey_Ediaeresis

kOfxKey_Igrave

kOfxKey_Iacute

kOfxKey_Icircumflex

kOfxKey_Idiaeresis

kOfxKey_ETH

kOfxKey_Eth

kOfxKey_Ntilde

kOfxKey_Ograve

kOfxKey_Oacute

kOfxKey_Ocircumflex

kOfxKey_Otilde

kOfxKey_Odiaeresis

kOfxKey_multiply

kOfxKey_Ooblique

kOfxKey_Ugrave

kOfxKey_Uacute

kOfxKey_Ucircumflex

kOfxKey_Udiaeresis

kOfxKey_Yacute

kOfxKey_THORN

kOfxKey_ssharp

kOfxKey_agrave

kOfxKey_aacute

kOfxKey_acircumflex

kOfxKey_atilde

kOfxKey_adiaeresis

kOfxKey_aring

kOfxKey_ae

kOfxKey_ccedilla

k0fxKey_egrave

k0fxKey_eacute

k0fxKey_ecircumflex

k0fxKey_ediaeresis

k0fxKey_igrave

k0fxKey_iacute

k0fxKey_icircumflex

k0fxKey_idiaeresis

k0fxKey_eth

k0fxKey_ntilde

k0fxKey_ograve

k0fxKey_oacute

k0fxKey_ocircumflex

k0fxKey_otilde

k0fxKey_odiaeresis

k0fxKey_division

k0fxKey_oslash

k0fxKey_ugrave

k0fxKey_uacute

k0fxKey_ucircumflex

k0fxKey_udiaeresis

kOfxKey_yacute

kOfxKey_thorn

kOfxKey_ydiaeresis

File ofxMemory.h

This file contains the API for general purpose memory allocation from a host.

Defines

kOfxMemorySuite

Typedefs

typedef struct *OfxMemorySuiteV1* **OfxMemorySuiteV1**

The OFX suite that implements general purpose memory management.

Use this suite for ordinary memory management functions, where you would normally use malloc/free or new/delete on ordinary objects.

For images, you should use the memory allocation functions in the image effect suite, as many hosts have specific image memory pools.

Note: C++ plugin developers will need to redefine new and delete as skins ontop of this suite.

struct **OfxMemorySuiteV1**

#include <ofxMemory.h> The OFX suite that implements general purpose memory management.

Use this suite for ordinary memory management functions, where you would normally use malloc/free or new/delete on ordinary objects.

For images, you should use the memory allocation functions in the image effect suite, as many hosts have specific image memory pools.

Note: C++ plugin developers will need to redefine new and delete as skins ontop of this suite.

Public Members

OfxStatus (***memoryAlloc**)(void *handle, size_t nBytes, void **allocatedData)

Allocate memory.

- `handle` - effect instance to associate with this memory allocation, or NULL.
- `nBytes` number of bytes to allocate
- `allocatedData` pointer to the return value. Allocated memory will be alligned for any use.

This function has the host allocate memory using its own memory resources and returns that to the plugin.

Return

- *kOfxStatOK* the memory was sucessfully allocated
- *kOfxStatErrMemory* the request could not be met and no memory was allocated

OfxStatus (***memoryFree**)(void *allocatedData)

Frees memory.

- `allocatedData` pointer to memory previously returned by *OfxMemorySuiteV1::memoryAlloc*

This function frees any memory that was previously allocated via *OfxMemorySuiteV1::memoryAlloc*.

Return

- *kOfxStatOK* the memory was sucessfully freed
- *kOfxStatErrBadHandle* `allocatedData` was not a valid pointer returned by *OfxMemorySuiteV1::memoryAlloc*

File ofxMessage.h

This file contains the Host API for end user message communication.

Defines

kOfxMessageSuite

kOfxMessageFatal

String used to type fatal error messages.

Fatal error messages should only be posted by a plugin when it can no longer continue operation.

kOfxMessageError

String used to type error messages.

Ordinary error messages should be posted when there is an error in operation that is recoverable by user intervention.

kOfxMessageWarning

String used to type warning messages.

Warnings indicate states that allow for operations to proceed, but are not necessarily optimal.

kOfxMessageMessage

String used to type simple ordinary messages.

Ordinary messages simply convey information from the plugin directly to the user.

kOfxMessageLog

String used to type log messages.

Log messages are written out to a log and not to the end user.

kOfxMessageQuestion

String used to type yes/no messages.

The host is to enter a modal state which waits for the user to respond yes or no. The *OfxMessageSuiteV1::message* function which posted the message will only return after the user responds. When asking a question, the *OfxStatus* code returned by the message function will be,

- *kOfxStatReplyYes* - if the user replied 'yes' to the question
- *kOfxStatReplyNo* - if the user replied 'no' to the question
- some error code if an error was encountered

It is an error to post a question message if the plugin is not in an interactive session.

Typedefs

typedef struct *OfxMessageSuiteV1* **OfxMessageSuiteV1**

The OFX suite that allows a plug-in to pass messages back to a user. The V2 suite extends on this in a backwards compatible manner.

typedef struct *OfxMessageSuiteV2* **OfxMessageSuiteV2**

The OFX suite that allows a plug-in to pass messages back to a user.

This extends *OfxMessageSuiteV1*, and should be considered a replacement to version 1.

Note that this suite has been extended in backwards compatible manner, so that a host can return this struct for both V1 and V2.

struct **OfxMessageSuiteV1**

#include <ofxMessage.h> The OFX suite that allows a plug-in to pass messages back to a user. The V2 suite extends on this in a backwards compatible manner.

Public Members

OfxStatus (***message**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a message on the host, using printf style varargs.

- `handle` effect handle (descriptor or instance) the message should be associated with, may be NULL
- `messageType` string describing the kind of message to post, one of the `kOfxMessageType*` constants
- `messageId` plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- `format` printf style format string
- ... printf style varargs list to print

Return

- *kOfxStatOK* - if the message was successfully posted
- *kOfxStatReplyYes* - if the message was of type `kOfxMessageQuestion` and the user reply yes
- *kOfxStatReplyNo* - if the message was of type `kOfxMessageQuestion` and the user reply no
- *kOfxStatFailed* - if the message could not be posted for some reason

struct **OfxMessageSuiteV2**

#include <ofxMessage.h> The OFX suite that allows a plug-in to pass messages back to a user.

This extends *OfxMessageSuiteV1*, and should be considered a replacement to version 1.

Note that this suite has been extended in backwards compatible manner, so that a host can return this struct for both V1 and V2.

Public Members

OfxStatus (***message**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a transient message on the host, using printf style varargs. Same as the V1 message suite call.

- `handle` effect handle (descriptor or instance) the message should be associated with, may be null
- `messageType` string describing the kind of message to post, one of the `kOfxMessageType*` constants
- `messageId` plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- `format` printf style format string
- ... printf style varargs list to print

Return

- *kOfxStatOK* - if the message was successfully posted

- *kOfxStatReplyYes* - if the message was of type *kOfxMessageQuestion* and the user reply yes
- *kOfxStatReplyNo* - if the message was of type *kOfxMessageQuestion* and the user reply no
- *kOfxStatFailed* - if the message could not be posted for some reason

OfxStatus (***setPersistentMessage**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a persistent message on an effect, using printf style varargs, and set error states. New for V2 message suite.

- *handle* effect instance handle the message should be associated with, may NOT be null,
- *messageType* string describing the kind of message to post, should be one of...
 - *kOfxMessageError*
 - *kOfxMessageWarning*
 - *kOfxMessageMessage*
- *messageId* plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- *format* printf style format string
- ... printf style varargs list to print

Persistent messages are associated with an effect *handle* until explicitly cleared by an effect. So if an error message is posted the error state, and associated message will persist and be displayed on the effect appropriately. (eg: draw a node in red on a node based compositor and display the message when clicked on).

If *messageType* is error or warning, associated error states should be flagged on host applications. Posting an error message implies that the host cannot proceed, a warning allows the host to proceed, whilst a simple message should have no stop anything.

Return

- *kOfxStatOK* - if the message was successfully posted
- *kOfxStatErrBadHandle* - the handle was rubbish
- *kOfxStatFailed* - if the message could not be posted for some reason

OfxStatus (***clearPersistentMessage**)(void *handle)

Clears any persistent message on an effect handle that was set by *OfxMessageSuiteV2::setPersistentMessage*. New for V2 message suite.

- *handle* effect instance handle messages should be cleared from.
- *handle* effect handle (descriptor or instance)

Clearing a message will clear any associated error state.

Return

- *kOfxStatOK* - if the message was successfully cleared

- *kOfxStatErrBadHandle* - the handle was rubbish
- *kOfxStatFailed* - if the message could not be cleared for some reason

File ofxMultiThread.h

This file contains the Host Suite for threading

Defines

kOfxMultiThreadSuite

Typedefs

```
typedef struct OfxMutex *OfxMutexHandle
```

Mutex blind data handle.

```
void() OfxThreadFunctionV1 (unsigned int threadIndex, unsigned int threadMax,  
void *customArg)
```

The function type to be passed to the multi threading routines.

- **threadIndex** unique index of this thread, will be between 0 and **threadMax**
- **threadMax** total number of threads executing this function
- **customArg** the argument passed into **multiThread**

A function of this type is passed to *OfxMultiThreadSuiteV1::multiThread* to be launched in multiple threads.

```
typedef struct OfxMultiThreadSuiteV1 OfxMultiThreadSuiteV1
```

OFX suite that provides simple SMP style multi-processing.

```
struct OfxMultiThreadSuiteV1
```

```
#include <ofxMultiThread.h> OFX suite that provides simple SMP style multi-processing.
```

Public Members

```
OfxStatus (*multiThread)(OfxThreadFunctionV1 func, unsigned int nThreads, void *customArg)
```

Function to spawn SMP threads.

- **func** function to call in each thread.
- **nThreads** number of threads to launch
- **customArg** parameter to pass to **customArg** of **func** in each thread.

This function will spawn `nThreads` separate threads of computation (typically one per CPU) to allow something to perform symmetric multi processing. Each thread will call 'func' passing in the index of the thread and the number of threads actually launched.

`multiThread` will not return until all the spawned threads have returned. It is up to the host how it waits for all the threads to return (busy wait, blocking, whatever).

`nThreads` can be more than the value returned by `multiThreadNumCPUs`, however the threads will be limited to the number of CPUs returned by `multiThreadNumCPUs`.

This function cannot be called recursively.

Return

- *kOfxStatOK*, the function `func` has executed and returned successfully
- *kOfxStatFailed*, the threading function failed to launch
- *kOfxStatErrExists*, failed in an attempt to call `multiThread` recursively,

OfxStatus (***multiThreadNumCPUs**)(unsigned int *nCPUs)

Function which indicates the number of CPUs available for SMP processing.

- `nCPUs` pointer to an integer where the result is returned

This value may be less than the actual number of CPUs on a machine, as the host may reserve other CPUs for itself.

Return

- *kOfxStatOK*, all was OK and the maximum number of threads is in `nThreads`.
- *kOfxStatFailed*, the function failed to get the number of CPUs

OfxStatus (***multiThreadIndex**)(unsigned int *threadIndex)

Function which indicates the index of the current thread.

- `threadIndex` pointer to an integer where the result is returned

This function returns the thread index, which is the same as the *threadIndex* argument passed to the *OfxThreadFunctionV1*.

If there are no threads currently spawned, then this function will set `threadIndex` to 0

Return

- *kOfxStatOK*, all was OK and the maximum number of threads is in `nThreads`.
- *kOfxStatFailed*, the function failed to return an index

int (***multiThreadIsSpawnedThread**)(void)

Function to enquire if the calling thread was spawned by `multiThread`.

Return

- 0 if the thread is not one spawned by `multiThread`
- 1 if the thread was spawned by `multiThread`

OfxStatus (***mutexCreate**)(*OfxMutexHandle* *mutex, int lockCount)

Create a mutex.

- `mutex` where the new handle is returned
- `count` initial lock count on the mutex. This can be negative.

Creates a new mutex with `lockCount` locks on the mutex initially set.

Return

- `kOfxStatOK` - mutex is now valid and ready to go

OfxStatus (***mutexDestroy**)(const *OfxMutexHandle* mutex)

Destroy a mutex.

Destroys a mutex initially created by `mutexCreate`.

Return

- `kOfxStatOK` - if it destroyed the mutex
- `kOfxStatErrBadHandle` - if the handle was bad

OfxStatus (***mutexLock**)(const *OfxMutexHandle* mutex)

Blocking lock on the mutex.

This tries to lock a mutex and blocks the thread it is in until the lock succeeds.

A successful lock causes the mutex's lock count to be increased by one and to block any other calls to lock the mutex until it is unlocked.

Return

- `kOfxStatOK` - if it got the lock
- `kOfxStatErrBadHandle` - if the handle was bad

OfxStatus (***mutexUnLock**)(const *OfxMutexHandle* mutex)

Unlock the mutex.

This unlocks a mutex. Unlocking a mutex decreases its lock count by one.

Return

- `kOfxStatOK` if it released the lock
- `kOfxStatErrBadHandle` if the handle was bad

OfxStatus (***mutexTryLock**)(const *OfxMutexHandle* mutex)

Non blocking attempt to lock the mutex.

This attempts to lock a mutex, if it cannot, it returns and says so, rather than blocking.

A successful lock causes the mutex's lock count to be increased by one, if the lock did not succeed, the call returns immediately and the lock count remains unchanged.

Return

- `kOfxStatOK` - if it got the lock

- `kOfxStatFailed` - if it did not get the lock
- `kOfxStatErrBadHandle` - if the handle was bad

File `ofxOld.h`

Defines

`kOfxImageComponentYUVA`

String to label images with YUVA components —`ofxImageEffects.h`.

Deprecated:

- removed in v1.4. Note, this has been deprecated in v1.3

`kOfxImageEffectPropInAnalysis`

Indicates whether an effect is performing an analysis pass. —`ofxImageEffects.h`.

- Type - int X 1
- Property Set - plugin instance (read/write)
- Default - to 0
- Valid Values - This must be one of 0 or 1

Deprecated:

- This feature has been deprecated - officially commented out v1.4.

`kOfxInteractPropViewportSize`

The size of an interact's OpenGL viewport — `ofxInteract.h`.

- Type - int X 2
- Property Set - read only property on the interact instance and in argument to all the interact actions.

Deprecated:

- V1.3: This property is the redundant and its use will be deprecated in future releases. V1.4: Removed

`kOfxParamDoubleTypeNormalisedX`

value for the `kOfxParamPropDoubleType` property, indicating a size normalised to the X dimension. See `kOfxParamPropDoubleType`. — `ofxParam.h`

Deprecated:

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeX` V1.4: Removed

kOfxParamDoubleTypeNormalisedY

value for the *kOfxParamPropDoubleType* property, indicating a size normalised to the Y dimension. See *kOfxParamPropDoubleType*. — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeY` V1.4: Removed

kOfxParamDoubleTypeNormalisedXAbsolute

value for the *kOfxParamPropDoubleType* property, indicating an absolute position normalised to the X dimension. See *kOfxParamPropDoubleType*. — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeXAbsolute` V1.4: Removed

kOfxParamDoubleTypeNormalisedYAbsolute

value for the *kOfxParamPropDoubleType* property, indicating an absolute position normalised to the Y dimension. See *kOfxParamPropDoubleType*. — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeYAbsolute` V1.4: Removed

kOfxParamDoubleTypeNormalisedXY

value for the *kOfxParamPropDoubleType* property, indicating normalisation to the X and Y dimension for 2D params. See *kOfxParamPropDoubleType*. — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeXY` V1.4: Removed

kOfxParamDoubleTypeNormalisedXYAbsolute

value for the *kOfxParamPropDoubleType* property, indicating normalisation to the X and Y dimension for a 2D param that can be interpreted as an absolute spatial position. See *kOfxParamPropDoubleType*. — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of *kOfxParamDoubleTypeXYAbsolute* V1.4: Removed

Typedefs

typedef struct *OfxYUVAColourB* **OfxYUVAColourB**

Defines an 8 bit per component YUVA pixel — *ofxPixels.h* Deprecated in 1.3, removed in 1.4.

typedef struct *OfxYUVAColourS* **OfxYUVAColourS**

Defines an 16 bit per component YUVA pixel — *ofxPixels.h*.

Deprecated:

- Deprecated in 1.3, removed in 1.4

typedef struct *OfxYUVAColourF* **OfxYUVAColourF**

Defines an floating point component YUVA pixel — *ofxPixels.h*.

Deprecated:

- Deprecated in 1.3, removed in 1.4

struct **OfxYUVAColourB**

#include <*ofxOld.h*> Defines an 8 bit per component YUVA pixel — *ofxPixels.h* Deprecated in 1.3, removed in 1.4.

Public Members

unsigned char **y**

unsigned char **u**

unsigned char **v**

unsigned char **a**

struct **OfxYUVAColourS**

#include <*ofxOld.h*> Defines an 16 bit per component YUVA pixel — *ofxPixels.h*.

Deprecated:

- Deprecated in 1.3, removed in 1.4

Public Members

unsigned short **y**

unsigned short **u**

unsigned short **v**

unsigned short **a**

struct **OfxYUVAColourF**

#include <ofxOld.h> Defines an floating point component YUVA pixel — *ofxPixels.h*.

Deprecated:

- Deprecated in 1.3, removed in 1.4

Public Members

float **y**

float **u**

float **v**

float **a**

File ofxOpenGLRender.h

Defines

_ofxOpenGLRender_h_

File ofxParam.h

This header contains the suite definition to manipulate host side parameters.

For more details go see [ParametersPage](#)

Defines

kOfxParameterSuite

string value to the *kOfxPropType* property for all parameters

kOfxTypeParameter

string value on the *kOfxPropType* property for all parameter definitions (ie: the handle returned in describe)

kOfxTypeParameterInstance

string value on the *kOfxPropType* property for all parameter instances

kOfxParamTypeInteger

String to identify a param as a single valued integer.

kOfxParamTypeDouble

String to identify a param as a Single valued floating point parameter

kOfxParamTypeBoolean

String to identify a param as a Single valued boolean parameter.

kOfxParamTypeChoice

String to identify a param as a Single valued, 'one-of-many' parameter.

kOfxParamTypeStrChoice

String to identify a param as a string-valued 'one-of-many' parameter.

Since

Version 1.5

kOfxParamTypeRGBA

String to identify a param as a Red, Green, Blue and Alpha colour parameter.

kOfxParamTypeRGB

String to identify a param as a Red, Green and Blue colour parameter.

kOfxParamTypeDouble2D

String to identify a param as a Two dimensional floating point parameter.

kOfxParamTypeInteger2D

String to identify a param as a Two dimensional integer point parameter.

kOfxParamTypeDouble3D

String to identify a param as a Three dimensional floating point parameter.

kOfxParamTypeInteger3D

String to identify a param as a Three dimensional integer parameter.

kOfxParamTypeString

String to identify a param as a String (UTF8) parameter.

kOfxParamTypeCustom

String to identify a param as a Plug-in defined parameter.

kOfxParamTypeGroup

String to identify a param as a Grouping parameter.

kOfxParamTypePage

String to identify a param as a page parameter.

kOfxParamTypePushButton

String to identify a param as a PushButton parameter.

kOfxParamHostPropSupportsCustomAnimation

Indicates if the host supports animation of custom parameters.

- Type - int X 1
- Property Set - host descriptor (read only)
- Value Values - 0 or 1

kOfxParamHostPropSupportsStringAnimation

Indicates if the host supports animation of string params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

kOfxParamHostPropSupportsBooleanAnimation

Indicates if the host supports animation of boolean params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

kOfxParamHostPropSupportsChoiceAnimation

Indicates if the host supports animation of choice params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

kOfxParamHostPropSupportsCustomInteract

Indicates if the host supports custom interacts for parameters.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

Currently custom interacts for parameters can only be drawn using OpenGL. APIs will be added later to support using the new Draw Suite.

kOfxParamHostPropMaxParameters

Indicates the maximum numbers of parameters available on the host.

- Type - int X 1
- Property Set - host descriptor (read only)

If set to -1 it implies unlimited number of parameters.

kOfxParamHostPropMaxPages

Indicates the maximum number of parameter pages.

- Type - int X 1
- Property Set - host descriptor (read only)

If there is no limit to the number of pages on a host, set this to -1.

Hosts that do not support paged parameter layout should set this to zero.

kOfxParamHostPropPageRowColumnCount

This indicates the number of parameter rows and columns on a page.

- Type - int X 2
- Property Set - host descriptor (read only)

If the host has supports paged parameter layout, used dimension 0 as the number of columns per page and dimension 1 as the number of rows per page.

kOfxParamPageSkipRow

Pseudo parameter name used to skip a row in a page layout.

Passed as a value to the *kOfxParamPropPageChild* property.

See ParametersInterfacesPagedLayouts for more details.

kOfxParamPageSkipColumn

Pseudo parameter name used to skip a row in a page layout.

Passed as a value to the *kOfxParamPropPageChild* property.

See ParametersInterfacesPagedLayouts for more details.

kOfxParamPropInteractV1

Overrides the parameter's standard user interface with the given interact.

- Type - pointer X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - NULL
- Valid Values - must point to a OfxPluginEntryPoint

If set, the parameter's normal interface is replaced completely by the interact gui.

Currently custom interacts for parameters can only be drawn using OpenGL. APIs will be added later to support using the new Draw Suite.

kOfxParamPropInteractSize

The size of a parameter instance's custom interface in screen pixels.

- Type - double x 2
- Property Set - plugin parameter instance (read only)

This is set by a host to indicate the current size of a custom interface if the plug-in has one. If not this is set to (0,0).

kOfxParamPropInteractSizeAspect

The preferred aspect ratio of a parameter's custom interface.

- Type - double x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1.0
- Valid Values - greater than or equal to 0.0

If set to anything other than 0.0, the custom interface for this parameter will be of a size with this aspect ratio (x size/y size).

kOfxParamPropInteractMinimumSize

The minimum size of a parameter's custom interface, in screen pixels.

- Type - double x 2
- Property Set - plugin parameter descriptor (read/write) and instance (read only)

- Default - 10,10
- Valid Values - greater than (0, 0)

Any custom interface will not be less than this size.

kOfxParamPropInteractPreferredSize

The preferred size of a parameter's custom interface.

- Type - int x 2
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 10,10
- Valid Values - greater than (0, 0)

A host should attempt to set a parameter's custom interface on a parameter to be this size if possible, otherwise it will be of *kOfxParamPropInteractSizeAspect* aspect but larger than *kOfxParamPropInteractMinimumSize*.

kOfxParamPropType

The type of a parameter.

- Type - C string X 1
- Property Set - plugin parameter descriptor (read only) and instance (read only)

This string will be set to the type that the parameter was create with.

kOfxParamPropAnimates

Flags whether a parameter can animate.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

A plug-in uses this property to indicate if a parameter is able to animate.

kOfxParamPropCanUndo

Flags whether changes to a parameter should be put on the undo/redo stack.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

kOfxPropParamSetNeedsSyncing

States whether the plugin needs to resync its private data.

- Type - int X 1
- Property Set - param set instance (read/write)
- Default - 0
- Valid Values -
 - 0 - no need to sync
 - 1 - paramset is not synced

The plugin should set this flag to true whenever any internal state has not been flushed to the set of params.

The host will examine this property each time it does a copy or save operation on the instance. If it is set to 1, the host will call SyncPrivateData and then set it to zero before doing the copy/save. If it is set to 0, the host will assume that the param data correctly represents the private state, and will not call SyncPrivateData before copying/saving. If this property is not set, the host will always call SyncPrivateData before copying or saving the effect (as if the property were set to 1 — but the host will not create or modify the property).

kOfxParamPropIsAnimating

Flags whether a parameter is currently animating.

- Type - int x 1
- Property Set - plugin parameter instance (read only)
- Valid Values - 0 or 1

Set by a host on a parameter instance to indicate if the parameter has a non-constant value set on it. This can be as a consequence of animation or of scripting modifying the value, or of a parameter being connected to an expression in the host.

kOfxParamPropPluginMayWrite

Flags whether the plugin will attempt to set the value of a parameter in some callback or analysis pass.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 0
- Valid Values - 0 or 1

This is used to tell the host whether the plug-in is going to attempt to set the value of the parameter.

Deprecated:

- v1.4: deprecated - to be removed in 1.5

kOfxParamPropPersistent

Flags whether the value of a parameter should persist.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

This is used to tell the host whether the value of the parameter is important and should be save in any description of the plug-in.

kOfxParamPropEvaluateOnChange

Flags whether changing a parameter's value forces an evaluation (ie: render),.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write only)
- Default - 1
- Valid Values - 0 or 1

This is used to indicate if the value of a parameter has any affect on an effect's output, eg: the parameter may be purely for GUI purposes, and so changing its value should not trigger a re-render.

kOfxParamPropSecret

Flags whether a parameter should be exposed to a user,.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write)
- Default - 0
- Valid Values - 0 or 1

If secret, a parameter is not exposed to a user in any interface, but should otherwise behave as a normal parameter.

Secret params are typically used to hide important state detail that would otherwise be unintelligible to a user, for example the result of a statical analysis that might need many parameters to store.

kOfxParamPropScriptName

The value to be used as the id of the parameter in a host scripting language.

- Type - ASCII C string X 1,
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - the unique name the parameter was created with.
- Valid Values - ASCII string unique to all parameters in the plug-in.

Many hosts have a scripting language that they use to set values of parameters and more. If so, this is the name of a parameter in such scripts.

kOfxParamPropCacheInvalidation

Specifies how modifying the value of a param will affect any output of an effect over time.

- Type - C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - *kOfxParamInvalidateValueChange*
- Valid Values - This must be one of
 - *kOfxParamInvalidateValueChange*
 - *kOfxParamInvalidateValueChangeToEnd*
 - *kOfxParamInvalidateAll*

Imagine an effect with an animating parameter in a host that caches rendered output. Think of the what happens when you add a new key frame. -If the parameter represents something like an absolute position, the cache will only need to be invalidated for the range of frames that keyframe affects.

- If the parameter represents something like a speed which is integrated, the cache will be invalidated from the keyframe until the end of the clip.
- There are potentially other situations where the entire cache will need to be invalidated (though I can't think of one off the top of my head).

kOfxParamInvalidateValueChange

Used as a value for the *kOfxParamPropCacheInvalidation* property.

kOfxParamInvalidateValueChangeToEnd

Used as a value for the *kOfxParamPropCacheInvalidation* property.

kOfxParamInvalidateAll

Used as a value for the *kOfxParamPropCacheInvalidation* property.

kOfxParamPropHint

A hint to the user as to how the parameter is to be used.

- Type - UTF8 C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - ""

kOfxParamPropDefault

The default value of a parameter.

- Type - The type is dependant on the parameter type as is the dimension.
- Property Set - plugin parameter descriptor (read/write) and instance (read/write only),

- Default - 0 cast to the relevant type (or "" for strings and custom parameters)

The exact type and dimension is dependant on the type of the parameter. These are...

- *kOfxParamTypeInteger* - integer property of one dimension
- *kOfxParamTypeDouble* - double property of one dimension
- *kOfxParamTypeBoolean* - integer property of one dimension
- *kOfxParamTypeChoice* - integer property of one dimension
- *kOfxParamTypeStrChoice* - string property of one dimension
- *kOfxParamTypeRGBA* - double property of four dimensions
- *kOfxParamTypeRGB* - double property of three dimensions
- *kOfxParamTypeDouble2D* - double property of two dimensions
- *kOfxParamTypeInteger2D* - integer property of two dimensions
- *kOfxParamTypeDouble3D* - double property of three dimensions
- *kOfxParamTypeInteger3D* - integer property of three dimensions
- *kOfxParamTypeString* - string property of one dimension
- *kOfxParamTypeCustom* - string property of one dimension
- *kOfxParamTypeGroup* - does not have this property
- *kOfxParamTypePage* - does not have this property
- *kOfxParamTypePushButton* - does not have this property

kOfxParamPropDoubleType

Describes how the double parameter should be interpreted by a host.

- Type - C string X 1
- Default - *kOfxParamDoubleTypePlain*
- Property Set - 1D, 2D and 3D float plugin parameter descriptor (read/write) and instance (read only),
- Valid Values -This must be one of
 - *kOfxParamDoubleTypePlain* - parameter has no special interpretation,
 - *kOfxParamDoubleTypeAngle* - parameter is to be interpreted as an angle,
 - *kOfxParamDoubleTypeScale* - parameter is to be interpreted as a scale factor,
 - *kOfxParamDoubleTypeTime* - parameter represents a time value (1D only),
 - *kOfxParamDoubleTypeAbsoluteTime* - parameter represents an absolute time value (1D only),
 - *kOfxParamDoubleTypeX* - size wrt to the project's X dimension (1D only), in canonical coordinates,
 - *kOfxParamDoubleTypeXAbsolute* - absolute position on the X axis (1D only), in canonical coordinates,
 - *kOfxParamDoubleTypeY* - size wrt to the project's Y dimension(1D only), in canonical coordinates,
 - *kOfxParamDoubleTypeYAbsolute* - absolute position on the Y axis (1D only), in canonical coordinates,
 - *kOfxParamDoubleTypeXY* - size in 2D (2D only), in canonical coordinates,
 - *kOfxParamDoubleTypeXYAbsolute* - an absolute position on the image plane, in canonical coordinates.

Double parameters can be interpreted in several different ways, this property tells the host how to do so and thus gives hints as to the interface of the parameter.

kOfxParamDoubleTypePlain

value for the *kOfxParamPropDoubleType* property, indicating the parameter has no special interpretation and should be interpreted as a raw numeric value.

kOfxParamDoubleTypeScale

value for the *kOfxParamPropDoubleType* property, indicating the parameter is to be interpreted as a scale factor. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeAngle

value for the *kOfxParamPropDoubleTypeAngle* property, indicating the parameter is to be interpreted as an angle. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeTime

value for the *kOfxParamPropDoubleTypeAngle* property, indicating the parameter is to be interpreted as a time. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeAbsoluteTime

value for the *kOfxParamPropDoubleTypeAngle* property, indicating the parameter is to be interpreted as an absolute time from the start of the effect. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeX

value for the *kOfxParamPropDoubleType* property, indicating a size in canonical coords in the X dimension. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeY

value for the *kOfxParamPropDoubleType* property, indicating a size in canonical coords in the Y dimension. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeXAbsolute

value for the *kOfxParamPropDoubleType* property, indicating an absolute position in canonical coords in the X dimension. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeYAbsolute

value for the *kOfxParamPropDoubleType* property, indicating an absolute position in canonical coords in the Y dimension. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeXY

value for the *kOfxParamPropDoubleType* property, indicating a 2D size in canonical coords. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeXYAbsolute

value for the *kOfxParamPropDoubleType* property, indicating a 2D position in canonical coords. See *kOfxParamPropDoubleType*.

kOfxParamPropDefaultCoordinateSystem

Describes in which coordinate system a spatial double parameter's default value is specified.

- Type - C string X 1
- Default - kOfxParamCoordinatesCanonical
- Property Set - Non normalised spatial double parameters, ie: any double param who's *kOfxParamProp-DoubleType* is set to one of...
 - kOfxParamDoubleTypeX
 - kOfxParamDoubleTypeXAbsolute
 - kOfxParamDoubleTypeY
 - kOfxParamDoubleTypeYAbsolute
 - kOfxParamDoubleTypeXY
 - kOfxParamDoubleTypeXYAbsolute
- Valid Values - This must be one of
 - kOfxParamCoordinatesCanonical - the default is in canonical coords
 - kOfxParamCoordinatesNormalised - the default is in normalised coordinates

This allows a spatial param to specify what its default is, so by saying normalised and “0.5” it would be in the ‘middle’, by saying canonical and 100 it would be at value 100 independent of the size of the image being applied to.

kOfxParamCoordinatesCanonical

Define the canonical coordinate system.

kOfxParamCoordinatesNormalised

Define the normalised coordinate system.

kOfxParamPropHasHostOverlayHandle

A flag to indicate if there is a host overlay UI handle for the given parameter.

- Type - int x 1
- Property Set - plugin parameter descriptor (read only)
- Valid Values - 0 or 1

If set to 1, then the host is flagging that there is some sort of native user overlay interface handle available for the given parameter.

kOfxParamPropUseHostOverlayHandle

A flag to indicate that the host should use a native UI overlay handle for the given parameter.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write only) and instance (read only)

- Default - 0
- Valid Values - 0 or 1

If set to 1, then a plugin is flagging to the host that the host should use a native UI overlay handle for the given parameter. A plugin can use this to keep a native look and feel for parameter handles. A plugin can use *kOfxParamPropHasHostOverlayHandle* to see if handles are available on the given parameter.

kOfxParamPropShowTimeMarker

Enables the display of a time marker on the host's time line to indicate the value of the absolute time param.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write)
- Default - 0
- Valid Values - 0 or 1

If a double parameter is has *kOfxParamPropDoubleType* set to *kOfxParamDoubleTypeAbsoluteTime*, then this indicates whether any marker should be made visible on the host's time line.

kOfxPluginPropParamPageOrder

Sets the parameter pages and order of pages.

- Type - C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - ""
- Valid Values - the names of any page param in the plugin

This property sets the preferred order of parameter pages on a host. If this is never set, the preferred order is the order the parameters were declared in.

kOfxParamPropPageChild

The names of the parameters included in a page parameter.

- Type - C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - ""
- Valid Values - the names of any parameter that is not a group or page, as well as *kOfxParamPageSkipRow* and *kOfxParamPageSkipColumn*

This is a property on parameters of type *kOfxParamTypePage*, and tells the page what parameters it contains. The parameters are added to the page from the top left, filling in columns as we go. The two pseudo param names *kOfxParamPageSkipRow* and *kOfxParamPageSkipColumn* are used to control layout.

Note parameters can appear in more than one page.

kOfxParamPropParent

The name of a parameter's parent group.

- Type - C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - "", which implies the "root" of the hierarchy,
- Valid Values - the name of a parameter with type of *kOfxParamTypeGroup*

Hosts that have hierarchical layouts of their params use this to recursively group parameter.

By default parameters are added in order of declaration to the 'root' hierarchy. This property is used to reparent params to a predefined param of type *kOfxParamTypeGroup*.

kOfxParamPropGroupOpen

Whether the initial state of a group is open or closed in a hierarchical layout.

- Type - int X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

This is a property on parameters of type *kOfxParamTypeGroup*, and tells the group whether it should be open or closed by default.

kOfxParamPropEnabled

Used to enable a parameter in the user interface.

- Type - int X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - 1
- Valid Values - 0 or 1

When set to 0 a user should not be able to modify the value of the parameter. Note that the plug-in itself can still change the value of a disabled parameter.

kOfxParamPropDataPtr

A private data pointer that the plug-in can store its own data behind.

- Type - pointer X 1
- Property Set - plugin parameter instance (read/write),
- Default - NULL

This data pointer is unique to each parameter instance, so two instances of the same parameter do not share the same data pointer. Use it to hang any needed private data structures.

kOfxParamPropChoiceOption

Set options of a choice parameter.

- Type - UTF8 C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the property is empty with no options set.

This property contains the set of options that will be presented to a user from a choice parameter. See `ParametersChoice` for more details.

kOfxParamPropChoiceOrder

Set values the host should store for a choice parameter.

- Type - int X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - Zero-based ordinal list of same length as `OfxParamPropChoiceOption`

This property specifies the order in which the options are presented. See “Choice Parameters” for more details. This property is optional; if not set, the host will present the options in their natural order.

This property is useful when changing order of choice param options, or adding new options in the middle, in a new version of the plugin.

```
Plugin v1:
Option = {"OptA", "OptB", "OptC"}
Order = {1, 2, 3}

Plugin v2:
// will be shown as OptA / OptB / NewOpt / OptC
Option = {"OptA", "OptB", "OptC", NewOpt"}
Order = {1, 2, 4, 3}
```

Note that this only affects the host UI’s display order; the project still stores the index of the selected option as always. Plugins should never reorder existing options if they desire backward compatibility.

Values may be arbitrary 32-bit integers. Behavior is undefined if the same value occurs twice in the list; plugins should not do that.

Since
Version 1.5

kOfxParamPropChoiceEnum

Set a enumeration string in a `StrChoice` (string-valued choice) parameter.

- Type - UTF8 C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the property is empty with no options set.

This property contains the set of enumeration strings stored by the host in the project corresponding to the options that will be presented to a user from a StrChoice parameter. See ParametersChoice for more details.

Since

Version 1.5

kOfxParamHostPropSupportsStrChoiceAnimation

Indicates if the host supports animation of string choice params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

Since

Version 1.5

kOfxParamHostPropSupportsStrChoice

Indicates if the host supports the StrChoice param type.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

Since

Version 1.5

kOfxParamPropMin

The minimum value for a numeric parameter.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the smallest possible value corresponding to the parameter type (eg: INT_MIN for an integer, -DBL_MAX for a double parameter)

Setting this will also reset *kOfxParamPropDisplayMin*.

kOfxParamPropMax

The maximum value for a numeric parameter.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),

- Default - the largest possible value corresponding to the parameter type (eg: INT_MAX for an integer, DBL_MAX for a double parameter)

Setting this will also reset `;;kOfxParamPropDisplayMax`.

kOfxParamPropDisplayMin

The minimum value for a numeric parameter on any user interface.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the smallest possible value corresponding to the parameter type (eg: INT_MIN for an integer, -DBL_MAX for a double parameter)

If a user interface represents a parameter with a slider or similar, this should be the minimum bound on that slider.

kOfxParamPropDisplayMax

The maximum value for a numeric parameter on any user interface.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the largest possible value corresponding to the parameter type (eg: INT_MAX for an integer, DBL_MAX for a double parameter)

If a user interface represents a parameter with a slider or similar, this should be the maximum bound on that slider.

kOfxParamPropIncrement

The granularity of a slider used to represent a numeric parameter.

- Type - double X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - 1
- Valid Values - any greater than 0.

This value is always in canonical coordinates for double parameters that are normalised.

kOfxParamPropDigits

How many digits after a decimal point to display for a double param in a GUI.

- Type - int X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - 2
- Valid Values - any greater than 0.

This applies to double params of any dimension.

kOfxParamPropDimensionLabel

Label for individual dimensions on a multidimensional numeric parameter.

- Type - UTF8 C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - “x”, “y” and “z”
- Valid Values - any

Use this on 2D and 3D double and integer parameters to change the label on an individual dimension in any GUI for that parameter.

kOfxParamPropIsAutoKeying

Will a value change on the parameter add automatic keyframes.

- Type - int X 1
- Property Set - plugin parameter instance (read only),
- Valid Values - 0 or 1

This is set by the host simply to indicate the state of the property.

kOfxParamPropCustomInterpCallbackV1

A pointer to a custom parameter’s interpolation function.

- Type - pointer X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - NULL
- Valid Values - must point to a *OfxCustomParamInterpFuncV1*

It is an error not to set this property in a custom parameter during a plugin’s define call if the custom parameter declares itself to be an animating parameter.

kOfxParamPropStringMode

Used to indicate the type of a string parameter.

- Type - C string X 1
- Property Set - plugin string parameter descriptor (read/write) and instance (read only),
- Default - *kOfxParamStringIsSingleLine*
- Valid Values - This must be one of the following
 - *kOfxParamStringIsSingleLine*
 - *kOfxParamStringIsMultiLine*

- *kOfxParamStringIsFilePath*
- *kOfxParamStringIsDirectoryPath*
- *kOfxParamStringIsLabel*
- *kOfxParamStringIsRichTextFormat*

kOfxParamPropStringFilePathExists

Indicates string parameters of file or directory type need that file to exist already.

- Type - int X 1
- Property Set - plugin string parameter descriptor (read/write) and instance (read only),
- Default - 1
- Valid Values - 0 or 1

If set to 0, it implies the user can specify a new file name, not just a pre-existing one.

kOfxParamStringIsSingleLine

Used to set a string parameter to be single line, value to be passed to a *kOfxParamPropStringMode* property.

kOfxParamStringIsMultiLine

Used to set a string parameter to be multiple line, value to be passed to a *kOfxParamPropStringMode* property.

kOfxParamStringIsFilePath

Used to set a string parameter to be a file path, value to be passed to a *kOfxParamPropStringMode* property.

kOfxParamStringIsDirectoryPath

Used to set a string parameter to be a directory path, value to be passed to a *kOfxParamPropStringMode* property.

kOfxParamStringIsLabel

Use to set a string parameter to be a simple label, value to be passed to a *kOfxParamPropStringMode* property

kOfxParamStringIsRichTextFormat

String value on the *kOfxParamPropStringMode* property of a string parameter (added in 1.3)

kOfxParamPropCustomValue

Used by interpolating custom parameters to get and set interpolated values.

- Type - C string X 1 or 2

This property is on the *inArgs* property and *outArgs* property of a *OfxCustomParamInterpFuncVI* and in both cases contains the encoded value of a custom parameter. As an *inArgs* property it will have two values, being the two keyframes to interpolate. As an *outArgs* property it will have a single value and the plugin should fill this with the encoded interpolated value of the parameter.

kOfxParamPropInterpolationTime

Used by interpolating custom parameters to indicate the time a key occurs at.

- Type - double X 2
- Property Set - inArgs parameter of a *OfxCustomParamInterpFuncV1* (read only)

The two values indicate the absolute times the surrounding keyframes occur at. The keyframes are encoded in a *kOfxParamPropCustomValue* property.

kOfxParamPropInterpolationAmount

Property used by *OfxCustomParamInterpFuncV1* to indicate the amount of interpolation to perform.

- Type - double X 1
- Property Set - inArgs parameter of a *OfxCustomParamInterpFuncV1* (read only)
- Valid Values - from 0 to 1

This property indicates how far between the two *kOfxParamPropCustomValue* keys to interpolate.

Typedefs

```
typedef struct OfxParamStruct *OfxParamHandle
```

Blind declaration of an OFX param.

```
typedef struct OfxParamSetStruct *OfxParamSetHandle
```

Blind declaration of an OFX parameter set.

```
OfxStatus() OfxCustomParamInterpFuncV1 (OfxParamSetHandle instance,  
OfxPropertySetHandle inArgs, OfxPropertySetHandle outArgs)
```

Function prototype for custom parameter interpolation callback functions.

- **instance** the plugin instance that this parameter occurs in
- **inArgs** handle holding the following properties...
 - **kOfxPropName** - the name of the custom parameter to interpolate
 - **kOfxPropTime** - absolute time the interpolation is occurring at
 - **kOfxParamPropCustomValue** - string property that gives the value of the two keyframes to interpolate, in this case 2D
 - **kOfxParamPropInterpolationTime** - 2D double property that gives the time of the two keyframes we are interpolating
 - **kOfxParamPropInterpolationAmount** - 1D double property indicating how much to interpolate between the two keyframes
- **outArgs** handle holding the following properties to be set
 - **kOfxParamPropCustomValue** - the value of the interpolated custom parameter, in this case 1D

This function allows custom parameters to animate by performing interpolation between keys.

The plugin needs to parse the two strings encoding keyframes on either side of the time we need a value for. It should then interpolate a new value for it, encode it into a string and set the *kOfxParamPropCustomValue* property with this on the outArgs handle.

The interp value is a linear interpolation amount, however this may be derived from a cubic (or other) curve.

typedef struct *OfxParameterSuiteV1* **OfxParameterSuiteV1**

The OFX suite used to define and manipulate user visible parameters.

struct **OfxParameterSuiteV1**

#include <ofxParam.h> The OFX suite used to define and manipulate user visible parameters.

Keyframe Handling

These functions allow the plug-in to delete and get information about keyframes.

To set keyframes, use *paramSetValueAtTime()*.

paramGetKeyTime and *paramGetKeyIndex* use indices to refer to keyframes. Keyframes are stored by the host in increasing time order, so $\text{time}(\text{kf}[i]) < \text{time}(\text{kf}[i+1])$. Keyframe indices will change whenever keyframes are added, deleted, or moved in time, whether by the host or by the plug-in. They may vary between actions if the user changes a keyframe. The keyframe indices will not change within a single action.

OfxStatus (***paramGetNumKeys**)(*OfxParamHandle* paramHandle, unsigned int *numberOfKeys)

Returns the number of keyframes in the parameter.

- *paramHandle* parameter handle to interrogate
- *numberOfKeys* pointer to integer where the return value is placed

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Returns the number of keyframes in the parameter.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetKeyTime**)(*OfxParamHandle* paramHandle, unsigned int nthKey, *OfxTime* *time)

Returns the time of the nth key.

- *paramHandle* parameter handle to interrogate
- *nthKey* which key to ask about (0 to *paramGetNumKeys* - 1), ordered by time
- *time* pointer to *OfxTime* where the return value is placed

Return

- *kOfxStatOK* - all was OK

- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrBadIndex* - the nthKey does not exist

OfxStatus (***paramGetKeyIndex**)(*OfxParamHandle* paramHandle, *OfxTime* time, int direction, int *index)

Finds the index of a keyframe at/before/after a specified time.

- paramHandle parameter handle to search
- time what time to search from
- direction
 - == 0 indicates search for a key at the indicated time (some small delta)
 - > 0 indicates search for the next key after the indicated time
 - < 0 indicates search for the previous key before the indicated time
- index pointer to an integer which in which the index is returned set to -1 if no key was found

Return

- *kOfxStatOK* - all was OK
- *kOfxStatFailed* - if the search failed to find a key
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramDeleteKey**)(*OfxParamHandle* paramHandle, *OfxTime* time)

Deletes a keyframe if one exists at the given time.

- paramHandle parameter handle to delete the key from
- time time at which a keyframe is

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrBadIndex* - no key at the given time

OfxStatus (***paramDeleteAllKeys**)(*OfxParamHandle* paramHandle)

Deletes all keyframes from a parameter.

- paramHandle parameter handle to delete the keys from
- name parameter to delete the keyframes from is

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Return

- *kOfxStatOK* - all was OK

- *kOfxStatErrBadHandle* - if the parameter handle was invalid

Public Members

OfxStatus (***paramDefine**)(*OfxParamSetHandle* paramSet, const char *paramType, const char *name, *OfxPropertySetHandle* *propertySet)

Defines a new parameter of the given type in a describe action.

- paramSet handle to the parameter set descriptor that will hold this parameter
- paramType type of the parameter to create, one of the *kOfxParamType** #defines
- name unique name of the parameter
- propertySet if not null, a pointer to the parameter descriptor's property set will be placed here.

This function defines a parameter in a parameter set and returns a property set which is used to describe that parameter.

This function does not actually create a parameter, it only says that one should exist in any subsequent instances. To fetch an parameter instance *paramGetHandle* must be called on an instance.

This function can always be called in one of a plug-in's "describe" functions which defines the parameter sets common to all instances of a plugin.

Return

- *kOfxStatOK* - the parameter was created correctly
- *kOfxStatErrBadHandle* - if the plugin handle was invalid
- *kOfxStatErrExists* - if a parameter of that name exists already in this plugin
- *kOfxStatErrUnknown* - if the type is unknown
- *kOfxStatErrUnsupported* - if the type is known but unsupported

OfxStatus (***paramGetHandle**)(*OfxParamSetHandle* paramSet, const char *name, *OfxParamHandle* *param, *OfxPropertySetHandle* *propertySet)

Retrieves the handle for a parameter in a given parameter set.

- paramSet instance of the plug-in to fetch the property handle from
- name parameter to ask about
- param pointer to a param handle, the value is returned here
- propertySet if not null, a pointer to the parameter's property set will be placed here.

Parameter handles retrieved from an instance are always distinct in each instance. The parameter handle is valid for the life-time of the instance. Parameter handles in instances are distinct from parameter handles in plugins. You cannot call this in a plugin's describe function, as it needs an instance to work on.

Return

- *kOfxStatOK* - the parameter was found and returned
- *kOfxStatErrBadHandle* - if the plugin handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramSetGetPropertySet**)(*OfiParamSetHandle* paramSet, *OfxPropertySetHandle* *propHandle)

Retrieves the property set handle for the given parameter set.

- paramSet parameter set to get the property set for
- propHandle pointer to a the property set handle, value is returned here

Note: The property handle belonging to a parameter set is the same as the property handle belonging to the plugin instance.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramGetPropertySet**)(*OfiParamHandle* param, *OfxPropertySetHandle* *propHandle)

Retrieves the property set handle for the given parameter.

- param parameter to get the property set for
- propHandle pointer to a the property set handle, value is returned here

The property handle is valid for the lifetime of the parameter, which is the lifetime of the instance that owns the parameter

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramGetValue**)(*OfiParamHandle* paramHandle, ...)

Gets the current value of a parameter,.

- paramHandle parameter handle to fetch value from
- ... one or more pointers to variables of the relevant type to hold the parameter's value

This gets the current value of a parameter. The varargs ... argument needs to be pointer to C variables of the relevant type for this parameter. Note that params with multiple values (eg Colour) take multiple args here. For example...

```
OfiParamHandle myDoubleParam, *myColourParam;
ofxHost->paramGetHandle(instance, "myDoubleParam", &myDoubleParam);
double myDoubleValue;
ofxHost->paramGetValue(myDoubleParam, &myDoubleValue);
```

(continues on next page)

(continued from previous page)

```

ofxHost->paramGetHandle(instance, "myColourParam", &myColourParam);
double myR, myG, myB;
ofxHost->paramGetValue(myColourParam, &myR, &myG, &myB);

```

Note: `paramGetValue` should only be called from within a *kOfxActionInstanceChanged* or interact action and never from the render actions (which should always use `paramGetValueAtTime`).

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetValueAtTime**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Gets the value of a parameter at a specific time.

- `paramHandle` parameter handle to fetch value from
- `time` at what point in time to look up the parameter
- ... one or more pointers to variables of the relevant type to hold the parameter's value

This gets the current value of a parameter. The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteVI::paramGetValue* for notes on the varargs list

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetDerivative**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Gets the derivative of a parameter at a specific time.

- `paramHandle` parameter handle to fetch value from
- `time` at what point in time to look up the parameter
- ... one or more pointers to variables of the relevant type to hold the parameter's derivative

This gets the derivative of the parameter at the indicated time.

The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteVI::paramGetValue* for notes on the varargs list.

Only double and colour params can have their derivatives found.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetIntegral**)(*OfxParamHandle* paramHandle, *OfxTime* time1, *OfxTime* time2, ...)

Gets the integral of a parameter over a specific time range,.

- **paramHandle** parameter handle to fetch integral from
- **time1** where to start evaluating the integral
- **time2** where to stop evaluating the integral
- ... one or more pointers to variables of the relevant type to hold the parameter's integral

This gets the integral of the parameter over the specified time range.

The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteV1::paramGetValue* for notes on the varargs list.

Only double and colour params can be integrated.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramSetValue**)(*OfxParamHandle* paramHandle, ...)

Sets the current value of a parameter.

- **paramHandle** parameter handle to set value in
- ... one or more variables of the relevant type to hold the parameter's value

This sets the current value of a parameter. The varargs ... argument needs to be values of the relevant type for this parameter. Note that params with multiple values (eg Colour) take multiple args here. For example...

```
ofxHost->paramSetValue(instance, "myDoubleParam", double(10));
ofxHost->paramSetValue(instance, "myColourParam", double(pix.r), double(pix.
↪g), double(pix.b));
```

Note: `paramSetValue` should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramSetValueAtTime**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Keyframes the value of a parameter at a specific time.

- **paramHandle** parameter handle to set value in
- **time** at what point in time to set the keyframe

- ... one or more variables of the relevant type to hold the parameter's value

This sets a keyframe in the parameter at the indicated time to have the indicated value. The varargs ... argument needs to be values of the relevant type for this parameter. See the note on *OfxParameterSuiteV1::paramSetValue* for more detail

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Note: `paramSetValueAtTime` should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramCopy**)(*OfxParamHandle* paramTo, *OfxParamHandle* paramFrom, *OfxTime* dstOffset, const *OfxRangeD* *frameRange)

Copies one parameter to another, including any animation etc...

- `paramTo` parameter to set
- `paramFrom` parameter to copy from
- `dstOffset` temporal offset to apply to keys when writing to the `paramTo`
- `frameRange` if `paramFrom` has animation, and `frameRange` is not null, only this range of keys will be copied

This copies the value of `paramFrom` to `paramTo`, including any animation it may have. All the previous values in `paramTo` will be lost.

To choose all animation in `paramFrom` set `frameRange` to [0, 0]

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Pre

- Both parameters must be of the same type.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramEditBegin**)(*OfxParamSetHandle* paramSet, const char *name)

Used to group any parameter changes for undo/redo purposes.

- `paramSet` the parameter set in which this is happening
- `name` label to attach to any undo/redo string UTF8

If a plugin calls `paramSetValue/paramSetValueAtTime` on one or more parameters, either from custom GUI interaction or some analysis of imagery etc.. this is used to indicate the start of a set of a parameter changes that should be considered part of a single undo/redo block.

See also *OfxParameterSuiteV1::paramEditEnd*

Note: `paramEditBegin` should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the instance handle was invalid

OfxStatus (***paramEditEnd**)(*OfxParamSetHandle* paramSet)

Used to group any parameter changes for undo/redo purposes.

- `paramSet` parameter set in which this is happening

If a plugin calls `paramSetValue/paramSetValueAtTime` on one or more parameters, either from custom GUI interaction or some analysis of imagery etc.. this is used to indicate the end of a set of parameter changes that should be considered part of a single undo/redo block

See also *OfxParameterSuiteV1::paramEditBegin*

Note: `paramEditEnd` should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the instance handle was invalid

File `ofxParametricParam.h`

This header file defines the optional OFX extension to define and manipulate parametric parameters.

Defines

kOfxParametricParameterSuite

string value to the *kOfxPropType* property for all parameters

kOfxParamTypeParametric

String to identify a param as a single valued integer.

kOfxParamPropParametricDimension

The dimension of a parametric param.

- Type - int X 1
- Property Set - parametric param descriptor (read/write) and instance (read only)
- default - 1
- Value Values - greater than 0

This indicates the dimension of the parametric param.

kOfxParamPropParametricUIColour

The colour of parametric param curve interface in any UI.

- Type - double X N
- Property Set - parametric param descriptor (read/write) and instance (read only)
- default - unset,
- Value Values - three values for each dimension (see *kOfxParamPropParametricDimension*) being interpreted as R, G and B of the colour for each curve drawn in the UI.

This sets the colour of a parametric param curve drawn a host user interface. A colour triple is needed for each dimension of the oparametric param.

If not set, the host should generally draw these in white.

kOfxParamPropParametricInteractBackground

Interact entry point to draw the background of a parametric parameter.

- Type - pointer X 1
- Property Set - plug-in parametric parameter descriptor (read/write) and instance (read only),
- Default - NULL, which implies the host should draw its default background.

Defines a pointer to an interact which will be used to draw the background of a parametric parameter's user interface. None of the pen or keyboard actions can ever be called on the interact.

The openGL transform will be set so that it is an orthographic transform that maps directly to the 'parametric' space, so that 'x' represents the parametric position and 'y' represents the evaluated value.

kOfxParamHostPropSupportsParametricAnimation

Property on the host to indicate support for parametric parameter animation.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values
 - 0 indicating the host does not support animation of parametric params,

- 1 indicating the host does support animation of parametric params,

kOfxParamPropParametricRange

Property to indicate the min and max range of the parametric input value.

- Type - double X 2
- Property Set - parameter descriptor (read/write only), and instance (read only)
- Default Value - (0, 1)
- Valid Values - any pair of numbers so that the first is less than the second.

This controls the min and max values that the parameter will be evaluated at.

Typedefs

```
typedef struct OfxParametricParameterSuiteV1 OfxParametricParameterSuiteV1
```

The OFX suite used to define and manipulate ‘parametric’ parameters.

This is an optional suite.

Parametric parameters are in effect ‘functions’ a plug-in can ask a host to arbitrarily evaluate for some value ‘x’. A classic use case would be for constructing look-up tables, a plug-in would ask the host to evaluate one at multiple values from 0 to 1 and use that to fill an array.

A host would probably represent this to a user as a cubic curve in a standard curve editor interface, or possibly through scripting. The user would then use this to define the ‘shape’ of the parameter.

The evaluation of such params is not the same as animation, they are returning values based on some arbitrary argument orthogonal to time, so to evaluate such a param, you need to pass a parametric position and time.

Often, you would want such a parametric parameter to be multi-dimensional, for example, a colour look-up table might want three values, one for red, green and blue. Rather than declare three separate parametric parameters, it would be better to have one such parameter with multiple values in it.

The major complication with these parameters is how to allow a plug-in to set values, and defaults. The default default value of a parametric curve is to be an identity lookup. If a plugin wishes to set a different default value for a curve, it can use the suite to set key/value pairs on the *descriptor* of the param. When a new instance is made, it will have these curve values as a default.

```
struct OfxParametricParameterSuiteV1
```

#include <ofxParametricParam.h> The OFX suite used to define and manipulate ‘parametric’ parameters.

This is an optional suite.

Parametric parameters are in effect ‘functions’ a plug-in can ask a host to arbitrarily evaluate for some value ‘x’. A classic use case would be for constructing look-up tables, a plug-in would ask the host to evaluate one at multiple values from 0 to 1 and use that to fill an array.

A host would probably represent this to a user as a cubic curve in a standard curve editor interface, or possibly through scripting. The user would then use this to define the ‘shape’ of the parameter.

The evaluation of such params is not the same as animation, they are returning values based on some arbitrary argument orthogonal to time, so to evaluate such a param, you need to pass a parametric position and time.

Often, you would want such a parametric parameter to be multi-dimensional, for example, a colour look-up table might want three values, one for red, green and blue. Rather than declare three separate parametric parameters, it would be better to have one such parameter with multiple values in it.

The major complication with these parameters is how to allow a plug-in to set values, and defaults. The default default value of a parametric curve is to be an identity lookup. If a plugin wishes to set a different default value for a curve, it can use the suite to set key/value pairs on the *descriptor* of the param. When a new instance is made, it will have these curve values as a default.

Public Members

OfxStatus (***parametricParamGetValue**)(*OfxParamHandle* param, int curveIndex, *OfxTime* time, double parametricPosition, double *returnValue)

Evaluates a parametric parameter.

- param handle to the parametric parameter
- curveIndex which dimension to evaluate
- time the time to evaluate to the parametric param at
- parametricPosition the position to evaluate the parametric param at
- returnValue pointer to a double where a value is returned

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrBadIndex* - the curve index was invalid

OfxStatus (***parametricParamGetNControlPoints**)(*OfxParamHandle* param, int curveIndex, double time, int *returnValue)

Returns the number of control points in the parametric param.

- param handle to the parametric parameter
- curveIndex which dimension to check
- time the time to check
- returnValue pointer to an integer where the value is returned.

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrBadIndex* - the curve index was invalid

OfxStatus (***parametricParamGetNthControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, int nthCtl, double *key, double *value)

Returns the key/value pair of the nth control point.

- param handle to the parametric parameter
- curveIndex which dimension to check
- time the time to check
- nthCtl the nth control point to get the value of
- key pointer to a double where the key will be returned
- value pointer to a double where the value will be returned

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***parametricParamSetNthControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, int nthCtl, double key, double value, bool addAnimationKey)

Modifies an existing control point on a curve.

- param handle to the parametric parameter
- curveIndex which dimension to set
- time the time to set the value at
- nthCtl the control point to modify
- key key of the control point
- value value of the control point
- addAnimationKey if the param is an animatable, setting this to true will force an animation keyframe to be set as well as a curve key, otherwise if false, a key will only be added if the curve is already animating.

This modifies an existing control point. Note that by changing key, the order of the control point may be modified (as you may move it before or after another point). So be careful when iterating over a curves control points and you change a key.

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***parametricParamAddControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, double key, double value, bool addAnimationKey)

Adds a control point to the curve.

- `param` handle to the parametric parameter
- `curveIndex` which dimension to set
- `time` the time to set the value at
- `key` key of the control point
- `value` value of the control point
- `addAnimationKey` if the `param` is an animatable, setting this to true will force an animation keyframe to be set as well as a curve key, otherwise if false, a key will only be added if the curve is already animating.

This will add a new control point to the given dimension of a parametric parameter. If a key exists sufficiently close to 'key', then it will be set to the indicated control point.

Return

- `kOfxStatOK` - all was fine
- `kOfxStatErrBadHandle` - if the parameter handle was invalid
- `kOfxStatErrUnknown` - if the type is unknown

OfxStatus (***parametricParamDeleteControlPoint**)(*OfxParamHandle* param, int curveIndex, int nthCtl)

Deletes the nth control point from a parametric param.

- `param` handle to the parametric parameter
- `curveIndex` which dimension to delete
- `nthCtl` the control point to delete

OfxStatus (***parametricParamDeleteAllControlPoints**)(*OfxParamHandle* param, int curveIndex)

Delete all curve control points on the given param.

- `param` handle to the parametric parameter
- `curveIndex` which dimension to clear

File `ofxPixels.h`

Contains pixel struct definitions

Typedefs

typedef struct *OfxRGBAColourB* **OfxRGBAColourB**

Defines an 8 bit per component RGBA pixel.

typedef struct *OfxRGBAColourS* **OfxRGBAColourS**

Defines a 16 bit per component RGBA pixel.

typedef struct *OfxRGBAColourF* **OfxRGBAColourF**

Defines a floating point component RGBA pixel.

typedef struct *OfxRGBAColourD* **OfxRGBAColourD**

Defines a double precision floating point component RGBA pixel.

typedef struct *OfxRGBColourB* **OfxRGBColourB**

Defines an 8 bit per component RGB pixel.

typedef struct *OfxRGBColourS* **OfxRGBColourS**

Defines a 16 bit per component RGB pixel.

typedef struct *OfxRGBColourF* **OfxRGBColourF**

Defines a floating point component RGB pixel.

typedef struct *OfxRGBColourD* **OfxRGBColourD**

Defines a double precision floating point component RGB pixel.

struct **OfxRGBAColourB**

#include <ofxPixels.h> Defines an 8 bit per component RGBA pixel.

Public Members

unsigned char **r**

unsigned char **g**

unsigned char **b**

unsigned char **a**

struct **OfxRGBAColourS**

#include <ofxPixels.h> Defines a 16 bit per component RGBA pixel.

Public Members

unsigned short **r**

unsigned short **g**

unsigned short **b**

unsigned short **a**

struct **OfxRGBAColourF**

#include <ofxPixels.h> Defines a floating point component RGBA pixel.

Public Members

float **r**

float **g**

float **b**

float **a**

struct **OfxRGBAColourD**

#include <ofxPixels.h> Defines a double precision floating point component RGBA pixel.

Public Members

double **r**

double **g**

double **b**

double **a**

struct **OfxRGBColourB**

#include <ofxPixels.h> Defines an 8 bit per component RGB pixel.

Public Membersunsigned char **r**unsigned char **g**unsigned char **b**struct **OfxRGBColourS***#include* <ofxPixels.h> Defines a 16 bit per component RGB pixel.**Public Members**unsigned short **r**unsigned short **g**unsigned short **b**struct **OfxRGBColourF***#include* <ofxPixels.h> Defines a floating point component RGB pixel.**Public Members**float **r**float **g**float **b**struct **OfxRGBColourD***#include* <ofxPixels.h> Defines a double precision floating point component RGB pixel.**Public Members**double **r**double **g**double **b**

File ofxProgress.h

Defines

kOfxProgressSuite

suite for displaying a progress bar

Typedefs

typedef struct *OfxProgressSuiteV1* **OfxProgressSuiteV1**

A suite that provides progress feedback from a plugin to an application.

A plugin instance can initiate, update and close a progress indicator with this suite.

This is an optional suite in the Image Effect API.

API V1.4: Amends the documentation of progress suite V1 so that it is expected that it can be raised in a modal manner and have a “cancel” button when invoked in instanceChanged. Plugins that perform analysis post an appropriate message, raise the progress monitor in a modal manner and should poll to see if processing has been aborted. Any cancellation should be handled gracefully by the plugin (eg: reset analysis parameters to default values), clear allocated memory...

Many hosts already operate as described above. kOfxStatReplyNo should be returned to the plugin during progressUpdate when the user presses cancel.

Suite V2: Adds an ID that can be looked up for internationalisation and so on. When a new version is introduced, because plug-ins need to support old versions, and plug-in’s new releases are not necessary in synch with hosts (or users don’t immediately update), best practice is to support the 2 suite versions. That is, the plugin should check if V2 exists; if not then check if V1 exists. This way a graceful transition is guaranteed. So plugin should fetchSuite passing 2, (OfxProgressSuiteV2*) fetchSuite(mHost->mHost->host, kOfxProgressSuite,2); and if no success pass (OfxProgressSuiteV1*) fetchSuite(mHost->mHost->host, kOfxProgressSuite,1);

typedef struct *OfxProgressSuiteV2* **OfxProgressSuiteV2**

struct **OfxProgressSuiteV1**

#include <ofxProgress.h> A suite that provides progress feedback from a plugin to an application.

A plugin instance can initiate, update and close a progress indicator with this suite.

This is an optional suite in the Image Effect API.

API V1.4: Amends the documentation of progress suite V1 so that it is expected that it can be raised in a modal manner and have a “cancel” button when invoked in instanceChanged. Plugins that perform analysis post an appropriate message, raise the progress monitor in a modal manner and should poll to see if processing has been aborted. Any cancellation should be handled gracefully by the plugin (eg: reset analysis parameters to default values), clear allocated memory...

Many hosts already operate as described above. kOfxStatReplyNo should be returned to the plugin during progressUpdate when the user presses cancel.

Suite V2: Adds an ID that can be looked up for internationalisation and so on. When a new version is introduced, because plug-ins need to support old versions, and plug-in’s new releases are not necessary in synch with hosts (or users don’t immediately update), best practice is to support the 2 suite versions. That is, the plugin should check if V2 exists; if not then check if V1 exists. This way a graceful transition is guaranteed. So plugin should

fetchSuite passing 2, (OfxProgressSuiteV2*) fetchSuite(mHost->mHost->host, kOfxProgressSuite,2); and if no success pass (OfxProgressSuiteV1*) fetchSuite(mHost->mHost->host, kOfxProgressSuite,1);

Public Members

OfxStatus (***progressStart**)(void *effectInstance, const char *label)

Initiate a progress bar display.

Call this to initiate the display of a progress bar.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.
- *label* a text label to display in any message portion of the progress object's user interface. A UTF8 string.

Pre

- There is no currently ongoing progress display for this instance.

Return

- *kOfxStatOK* - the handle is now valid for use
- *kOfxStatFailed* - the progress object failed for some reason
- *kOfxStatErrBadHandle* - *effectInstance* was invalid

OfxStatus (***progressUpdate**)(void *effectInstance, double progress)

Indicate how much of the processing task has been completed and reports on any abort status.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.
- *progress* a number between 0.0 and 1.0 indicating what proportion of the current task has been processed.

Return

- *kOfxStatOK* - the progress object was successfully updated and the task should continue
- *kOfxStatReplyNo* - the progress object was successfully updated and the task should abort
- *kOfxStatErrBadHandle* - the progress handle was invalid,

OfxStatus (***progressEnd**)(void *effectInstance)

Signal that we are finished with the progress meter.

Call this when you are done with the progress meter and no longer need it displayed.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.

Post

- you can no longer call *progressUpdate* on the instance

Return

- *kOfxStatOK* - the progress object was successfully closed
- *kOfxStatErrBadHandle* - the progress handle was invalid,

struct **OfxProgressSuiteV2**

#include <ofxProgress.h>

Public Members

OfxStatus (***progressStart**)(void *effectInstance, const char *message, const char *messageid)

Initiate a progress bar display.

Call this to initiate the display of a progress bar.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.
- *message* a text label to display in any message portion of the progress object's user interface. A UTF8 string.
- *messageId* plugin-specified id to associate with this message. If overriding the message in an XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur. New in V2 of this suite.

Pre

- There is no currently ongoing progress display for this instance.

Return

- *kOfxStatOK* - the handle is now valid for use
- *kOfxStatFailed* - the progress object failed for some reason
- *kOfxStatErrBadHandle* - *effectInstance* was invalid

OfxStatus (***progressUpdate**)(void *effectInstance, double progress)

Indicate how much of the processing task has been completed and reports on any abort status.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.
- *progress* a number between 0.0 and 1.0 indicating what proportion of the current task has been processed.

Return

- *kOfxStatOK* - the progress object was successfully updated and the task should continue
- *kOfxStatReplyNo* - the progress object was successfully updated and the task should abort
- *kOfxStatErrBadHandle* - the progress handle was invalid,

OfxStatus (***progressEnd**)(void *effectInstance)

Signal that we are finished with the progress meter.

Call this when you are done with the progress meter and no longer need it displayed.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.

Post

- you can no longer call *progressUpdate* on the instance

Return

- *kOfxStatOK* - the progress object was successfully closed
- *kOfxStatErrBadHandle* - the progress handle was invalid,

File *ofxProperty.h*

Contains the API for manipulating generic properties. For more details see *PropertiesPage*.

Defines

kOfxPropertySuite

Typedefs

typedef struct *OfxPropertySuiteV1* **OfxPropertySuiteV1**

The OFX suite used to access properties on OFX objects.

struct **OfxPropertySuiteV1**

#include <*ofxProperty.h*> The OFX suite used to access properties on OFX objects.

Public Members

OfxStatus (***propSetPointer**)(*OfxPropertySetHandle* properties, const char *property, int index, void *value)

Set a single value in a pointer property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *index* for multidimensional properties and is dimension of the one we are setting
- *value* value of the property we are setting

Return

- *kOfxStatOK*

- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetString**)(*OfxPropertySetHandle* properties, const char *property, int index, const char *value)

Set a single value in a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` for multidimensional properties and is dimension of the one we are setting
- `value` value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetDouble**)(*OfxPropertySetHandle* properties, const char *property, int index, double value)

Set a single value in a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` for multidimensional properties and is dimension of the one we are setting
- `value` value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetInt**)(*OfxPropertySetHandle* properties, const char *property, int index, int value)

Set a single value in an int property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` for multidimensional properties and is dimension of the one we are setting
- `value` value of the property we are setting

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`
- `kOfxStatErrUnknown`
- `kOfxStatErrBadIndex`
- `kOfxStatErrValue`

`OfxStatus` (***propSetPointerN**)(`OfxPropertySetHandle` properties, const char *property, int count, void *const *value)

Set multiple values of the pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are setting in that property (ie: indices 0..count-1)
- `value` pointer to an array of property values

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`
- `kOfxStatErrUnknown`
- `kOfxStatErrBadIndex`
- `kOfxStatErrValue`

`OfxStatus` (***propSetStringN**)(`OfxPropertySetHandle` properties, const char *property, int count, const char *const *value)

Set multiple values of a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are setting in that property (ie: indices 0..count-1)
- `value` pointer to an array of property values

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`

- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetDoubleN**)(*OfxPropertySetHandle* properties, const char *property, int count, const double *value)

Set multiple values of a double property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *count* number of values we are setting in that property (ie: indices 0..count-1)
- *value* pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetIntN**)(*OfxPropertySetHandle* properties, const char *property, int count, const int *value)

Set multiple values of an int property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *count* number of values we are setting in that property (ie: indices 0..count-1)
- *value* pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propGetPointer**)(*OfxPropertySetHandle* properties, const char *property, int index, void **value)

Get a single value from a pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`
- `kOfxStatErrUnknown`
- `kOfxStatErrBadIndex`

OfxStatus (***propGetString**)(*OfxPropertySetHandle* properties, const char *property, int index, char **value)

Get a single value of a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`
- `kOfxStatErrUnknown`
- `kOfxStatErrBadIndex`

OfxStatus (***propGetDouble**)(*OfxPropertySetHandle* properties, const char *property, int index, double *value)

Get a single value of a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

See the note ArchitectureStrings for how to deal with strings.

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`
- `kOfxStatErrUnknown`

- *kOfxStatErrBadIndex*

OfxStatus (***propGetInt**)(*OfxPropertySetHandle* properties, const char *property, int index, int *value)

Get a single value of an int property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetPointerN**)(*OfxPropertySetHandle* properties, const char *property, int count, void **value)

Get multiple values of a pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indicies 0..count-1)
- `value` pointer to an array of where we will return the property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetStringN**)(*OfxPropertySetHandle* properties, const char *property, int count, char **value)

Get multiple values of a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indicies 0..count-1)
- `value` pointer to an array of where we will return the property values

See the note ArchitectureStrings for how to deal with strings.

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetDoubleN**)(*OfxPropertySetHandle* properties, const char *property, int count, double *value)

Get multiple values of a double property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *count* number of values we are getting of that property (ie: indices 0..count-1)
- *value* pointer to an array of where we will return the property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetIntN**)(*OfxPropertySetHandle* properties, const char *property, int count, int *value)

Get multiple values of an int property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *count* number of values we are getting of that property (ie: indices 0..count-1)
- *value* pointer to an array of where we will return the property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propReset**)(*OfxPropertySetHandle* properties, const char *property)

Resets all dimensions of a property to its default value.

- *properties* handle of the thing holding the property

- property string labelling the property we are resetting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*

OfxStatus (***propGetDimension**)(*OfxPropertySetHandle* properties, const char *property, int *count)

Gets the dimension of the property.

- properties handle of the thing holding the property
- property string labelling the property we are resetting
- count pointer to an integer where the value is returned

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*

File ofxTimeLine.h

Defines

kOfxTimeLineSuite

Name of the time line suite.

Typedefs

typedef struct *OfxTimeLineSuiteV1* **OfxTimeLineSuiteV1**

Suite to control timelines.

This suite is used to enquire and control a timeline associated with a plug-in instance.

This is an optional suite in the Image Effect API.

struct **OfxTimeLineSuiteV1**

#include <ofxTimeLine.h> Suite to control timelines.

This suite is used to enquire and control a timeline associated with a plug-in instance.

This is an optional suite in the Image Effect API.

Public Members

OfxStatus (***getTime**)(void *instance, double *time)

Get the time value of the timeline that is controlling to the indicated effect.

- *instance* is the instance of the effect changing the timeline, cast to a void *
- *time* pointer through which the timeline value should be returned

This function returns the current time value of the timeline associated with the effect instance.

Return

- *kOfxStatOK* - the time enquiry was successful
- *kOfxStatFailed* - the enquiry failed for some host specific reason
- *kOfxStatErrBadHandle* - the effect handle was invalid

OfxStatus (***gotoTime**)(void *instance, double time)

Move the timeline control to the indicated time.

- *instance* is the instance of the effect changing the timeline, cast to a void *
- *time* is the time to change the timeline to. This is in the temporal coordinate system of the effect.

This function moves the timeline to the indicated frame and returns. Any side effects of the timeline change are also triggered and completed before this returns (for example instance changed actions and renders if the output of the effect is being viewed).

Return

- *kOfxStatOK* - the time was changed successfully, will all side effects if the change completed
- *kOfxStatFailed* - the change failed for some host specific reason
- *kOfxStatErrBadHandle* - the effect handle was invalid
- *kOfxStatErrValue* - the time was an illegal value

OfxStatus (***getTimeBounds**)(void *instance, double *firstTime, double *lastTime)

Get the current bounds on a timeline.

- *instance* is the instance of the effect changing the timeline, cast to a void *
- *firstTime* is the first time on the timeline. This is in the temporal coordinate system of the effect.
- *lastTime* is last time on the timeline. This is in the temporal coordinate system of the effect.

This function

Return

- *kOfxStatOK* - the time enquiry was successful
- *kOfxStatFailed* - the enquiry failed for some host specific reason
- *kOfxStatErrBadHandle* - the effect handle was invalid

1.21.2 Struct list

Struct `OfxDialogSuiteV1`

struct `OfxDialogSuiteV1`

Public Members

OfxStatus (***RequestDialog**)(void *user_data)

Request the host to send a `kOfxActionDialog` to the plugin from its UI thread.

Pre

- user_data: A pointer to any user data

Post

Return

- *kOfxStatOK* - The host has queued the request and will send an 'OfxActionDialog'
- *kOfxStatFailed* - The host has no provision for this or can not deal with it currently.

OfxStatus (***NotifyRedrawPending**)(void)

Inform the host of redraw event so it can redraw itself. If the host runs fullscreen in OpenGL, it would otherwise not receive redraw event when a dialog in front would catch all events.

Pre

Post

Return

- *kOfxStatReplyDefault*

Struct `OfxDrawSuiteV1`

struct `OfxDrawSuiteV1`

OFX suite that allows an effect to draw to a host-defined display context.

Public Members

OfxStatus (***getColor**)(*OfxDrawContextHandle* context, *OfxStandardColour* std_colour, *OfxRGBAColourF* *colour)

Retrieves the host's desired draw colour for.

- context draw context
- std_colour desired colour type
- colour returned RGBA colour

Return

- *kOfxStatOK* - the colour was returned
- *kOfxStatErrValue* - std_colour was invalid
- *kOfxStatFailed* - failure, e.g. if function is called outside kOfxInteractActionDraw

OfxStatus (***setColour**)(*OfxDrawContextHandle* context, const *OfxRGBAColourF* *colour)

Sets the colour for future drawing operations (lines, filled shapes and text)

- context draw context
- colour RGBA colour

The host should use “over” compositing when using a non-opaque colour.

Return

- *kOfxStatOK* - the colour was changed
- *kOfxStatFailed* - failure, e.g. if function is called outside kOfxInteractActionDraw

OfxStatus (***setLineWidth**)(*OfxDrawContextHandle* context, float width)

Sets the line width for future line drawing operations.

- context draw context
- width line width

Use width 0 for a single pixel line or non-zero for a smooth line of the desired width

The host should adjust for screen density.

Return

- *kOfxStatOK* - the width was changed
- *kOfxStatFailed* - failure, e.g. if function is called outside kOfxInteractActionDraw

OfxStatus (***setLineStipple**)(*OfxDrawContextHandle* context, *OfxDrawLineStipplePattern* pattern)

Sets the stipple pattern for future line drawing operations.

- context draw context
- pattern desired stipple pattern

Return

- *kOfxStatOK* - the pattern was changed
- *kOfxStatErrValue* - pattern was not valid
- *kOfxStatFailed* - failure, e.g. if function is called outside kOfxInteractActionDraw

OfxStatus (*draw)(*OfxDrawContextHandle* context, *OfxDrawPrimitive* primitive, const *OfxPointD* *points, int point_count)

Draws a primitive of the desired type.

- **context** draw context
- **primitive** desired primitive
- **points** array of points in the primitive
- **point_count** number of points in the array

kOfxDrawPrimitiveLines - like *GL_LINES*, n points draws n/2 separated lines *kOfxDrawPrimitiveLineStrip* - like *GL_LINE_STRIP*, n points draws n-1 connected lines *kOfxDrawPrimitiveLineLoop* - like *GL_LINE_LOOP*, n points draws n connected lines *kOfxDrawPrimitiveRectangle* - draws an axis-aligned filled rectangle defined by 2 opposite corner points *kOfxDrawPrimitivePolygon* - like *GL_POLYGON*, draws a filled n-sided polygon *kOfxDrawPrimitiveEllipse* - draws a axis-aligned elliptical line (not filled) within the rectangle defined by 2 opposite corner points

Return

- *kOfxStatOK* - the draw was completed
- *kOfxStatErrValue* - invalid primitive, or point_count not valid for primitive
- *kOfxStatFailed* - failure, e.g. if function is called outside *kOfxInteractActionDraw*

OfxStatus (*drawText)(*OfxDrawContextHandle* context, const char *text, const *OfxPointD* *pos, int alignment)

Draws text at the specified position.

- **context** draw context
- **text** text to draw (UTF-8 encoded)
- **pos** position at which to align the text
- **alignment** text alignment flags (see *kOfxDrawTextAlignment**)

The text font face and size are determined by the host.

Return

- *kOfxStatOK* - the text was drawn
- *kOfxStatErrValue* - text or pos were not defined
- *kOfxStatFailed* - failure, e.g. if function is called outside *kOfxInteractActionDraw*

Struct `OfxHost`

struct `OfxHost`

Generic host structure passed to `OfxPlugin::setHost` function.

This structure contains what is needed by a plug-in to bootstrap its connection to the host.

Public Members

`OfxPropertySetHandle` `host`

Global handle to the host. Extract relevant host properties from this. This pointer will be valid while the binary containing the plug-in is loaded.

const void `(*fetchSuite)(OfxPropertySetHandle host, const char *suiteName, int suiteVersion)`

The function which the plug-in uses to fetch suites from the host.

- `host` the host the suite is being fetched from this *must* be the `host` member of the `OfxHost` struct containing `fetchSuite`.
- `suiteName` ASCII string labelling the host supplied API
- `suiteVersion` version of that suite to fetch

Any API fetched will be valid while the binary containing the plug-in is loaded.

Repeated calls to `fetchSuite` with the same parameters will return the same pointer.

It is recommended that hosts should return the same host and suite pointers to all plugins in the same shared lib or bundle.

returns

- NULL if the API is unknown (either the api or the version requested),
- pointer to the relevant API if it was found

Struct `OfxImageEffectOpenGLRenderSuiteV1`

struct `OfxImageEffectOpenGLRenderSuiteV1`

OFX suite that provides image to texture conversion for OpenGL processing.

Public Members

`OfxStatus (*clipLoadTexture)(OfxImageClipHandle clip, OfxTime time, const char *format, const OfxRectD *region, OfxPropertySetHandle *textureHandle)`

loads an image from an OFX clip as a texture into OpenGL

- `clip` clip to load the image from
- `time` effect time to load the image from

- `format` requested texture format (As in none,byte,word,half,float, etc..) When set to NULL, the host decides the format based on the plug-in's `kOfxOpenGLPropPixelDepth` setting.
- `region` region of the image to load (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- `textureHandle` property set containing information about the texture

An image is fetched from a clip at the indicated time for the given region and loaded into an OpenGL texture. When a specific format is requested, the host ensures it gives the requested format. When the clip specified is the "Output" clip, the format is ignored and the host must bind the resulting texture as the current color buffer (render target). This may also be done prior to calling the `kOfxImageEffectActionRender` action. If the `region` parameter is set to non-NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set or is NULL, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped to the clip's Region of Definition. Information about the texture, including the texture index, is returned in the `textureHandle` argument. The properties on this handle will be...

- `kOfxImageEffectPropOpenGLTextureIndex`
- `kOfxImageEffectPropOpenGLTextureTarget`
- `kOfxImageEffectPropPixelDepth`
- `kOfxImageEffectPropComponents`
- `kOfxImageEffectPropPreMultiplication`
- `kOfxImageEffectPropRenderScale`
- `kOfxImagePropPixelAspectRatio`
- `kOfxImagePropBounds`
- `kOfxImagePropRegionOfDefinition`
- `kOfxImagePropRowBytes`
- `kOfxImagePropField`
- `kOfxImagePropUniqueIdentifier`

With the exception of the OpenGL specifics, these properties are the same as the properties in an image handle returned by `clipGetImage` in the image effect suite.

Note:

- this is the OpenGL equivalent of `clipGetImage` from `OfxImageEffectSuiteV1`
-

Pre

- clip was returned by `clipGetHandle`
- Format property in the texture handle

Post

- texture handle to be disposed of by `clipFreeTexture` before the action returns
- when the clip specified is the "Output" clip, the format is ignored and the host must bind the resulting texture as the current color buffer (render target). This may also be done prior to calling the render action.

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time and/or region, the plug-in should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - not enough OpenGL memory was available for the effect to load the texture. The plug-in should abort the GL render and return *kOfxStatErrMemory*, after which the host can decide to retry the operation with CPU based processing.

OfxStatus (*clipFreeTexture)(*OfxPropertySetHandle* textureHandle)

Releases the texture handle previously returned by clipLoadTexture.

For input clips, this also deletes the texture from OpenGL. This should also be called on the output clip; for the Output clip, it just releases the handle but does not delete the texture (since the host will need to read it).

Pre

- textureHandle was returned by clipGetImage

Post

- all operations on textureHandle will be invalid, and the OpenGL texture it referred to has been deleted (for source clips)

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - general failure for some reason,
- *kOfxStatErrBadHandle* - the image handle was invalid,

OfxStatus (*flushResources)()

Request the host to minimize its GPU resource load.

When a plug-in fails to allocate GPU resources, it can call this function to request the host to flush its GPU resources if it holds any. After the function the plug-in can try again to allocate resources which then might succeed if the host actually has released anything.

Pre

Post

- No changes to the plug-in GL state should have been made.

Return

- *kOfxStatOK* - the host has actually released some resources,
- *kOfxStatReplyDefault* - nothing the host could do..

Struct `OfxImageEffectSuiteV1`

struct **OfxImageEffectSuiteV1**

The OFX suite for image effects.

This suite provides the functions needed by a plugin to defined and use an image effect plugin.

Public Members

OfxStatus (***getPropertySet**)(*OfxImageEffectHandle* imageEffect, *OfxPropertySetHandle* *propHandle)

Retrieves the property set for the given image effect.

- `imageEffect` image effect to get the property set for
- `propHandle` pointer to a the property set pointer, value is returned here

The property handle is for the duration of the image effect handle.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***getParamSet**)(*OfxImageEffectHandle* imageEffect, *OfiParamSetHandle* *paramSet)

Retrieves the parameter set for the given image effect.

- `imageEffect` image effect to get the property set for
- `paramSet` pointer to a the parameter set, value is returned here

The param set handle is valid for the lifetime of the image effect handle.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***clipDefine**)(*OfxImageEffectHandle* imageEffect, const char *name, *OfxPropertySetHandle* *propertySet)

Define a clip to the effect.

- `pluginHandle` handle passed into ‘describeInContext’ action
- `name` unique name of the clip to define
- `propertySet` property handle for the clip descriptor will be returned here

This function defines a clip to a host, the returned property set is used to describe various aspects of the clip to the host. Note that this does not create a clip instance.

Pre

- we are inside the describe in context action.

Return

OfxStatus (*clipGetHandle)(*OfxImageEffectHandle* imageEffect, const char *name, *OfxImageClipHandle* *clip, *OfxPropertySetHandle* *propertySet)

Get the property handle of the named input clip in the given instance.

- imageEffect an instance handle to the plugin
- name name of the clip, previously used in a clip define call
- clip where to return the clip
- propertySet if not NULL, the descriptor handle for a parameter's property set will be placed here.

The propertySet will have the same value as would be returned by *OfxImageEffectSuiteV1::clipGetPropertySet*

This **return** a clip handle **for** the given instance, note that this will **not** be the same **as** the clip handle returned by clipDefine **and** will be distant to clip handles **in any** other instance of the plugin.

Not a valid call **in any** of the describe actions.

Pre

- create instance action called,
- name passed to clipDefine for this context,
- not inside describe or describe in context actions.

Post

- handle will be valid for the life time of the instance.

OfxStatus (*clipGetPropertySet)(*OfxImageClipHandle* clip, *OfxPropertySetHandle* *propHandle)

Retrieves the property set for a given clip.

- clip clip effect to get the property set for
- propHandle pointer to a the property set handle, value is returned here

The property handle is valid for the lifetime of the clip, which is generally the lifetime of the instance.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***clipGetImage**)(*OfxImageClipHandle* clip, *OfxTime* time, const *OfxRectD* *region, *OfxPropertySetHandle* *imageHandle)

Get a handle for an image in a clip at the indicated time and indicated region.

- `clip` clip to extract the image from
- `time` time to fetch the image at
- `region` region to fetch the image from (optional, set to NULL to get a ‘default’ region) this is in the CanonicalCoordinates.
- `imageHandle` property set containing the image’s data

An image is fetched from a clip at the indicated time for the given region and returned in the imageHandle.

If the *region* parameter is not set to NULL, then it will be clipped to the clip’s Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped the clip’s Region of Definition.

If clipGetImage is called twice with the same parameters, then two separate image handles will be returned, each of which must be release. The underlying implementation could share image data pointers and use reference counting to maintain them.

Pre

- clip was returned by clipGetHandle

Post

- image handle is only valid for the duration of the action clipGetImage is called in
- image handle to be disposed of by clipReleaseImage before the action returns

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time and/or region, the plugin should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - the host had not enough memory to complete the operation, plugin should abort whatever it was doing.

OfxStatus (***clipReleaseImage**)(*OfxPropertySetHandle* imageHandle)

Releases the image handle previously returned by clipGetImage.

Pre

- imageHandle was returned by clipGetImage

Post

- all operations on imageHandle will be invalid

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatErrBadHandle* - the image handle was invalid,

OfxStatus (***clipGetRegionOfDefinition**)(*OfxImageClipHandle* clip, *OfxTime* time, *OfxRectD* *bounds)

Returns the spatial region of definition of the clip at the given time.

- `clipHandle` clip to extract the image from
- `time` time to fetch the image at
- `region` region to fetch the image from (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- `imageHandle` handle where the image is returned

An image is fetched from a clip at the indicated time for the given region and returned in the `imageHandle`.

If the *region* parameter is not set to NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped the clip's Region of Definition.

Pre

- `clipHandle` was returned by `clipGetHandle`

Post

- `bounds` will be filled the RoD of the clip at the indicated time

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time, the plugin should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - the host had not enough memory to complete the operation, plugin should abort whatever it was doing.

int (***abort**)(*OfxImageEffectHandle* imageEffect)

Returns whether to abort processing or not.

- `imageEffect` instance of the image effect

A host may want to signal to a plugin that it should stop whatever rendering it is doing and start again. Generally this is done in interactive threads in response to users tweaking some parameter.

This function indicates whether a plugin should stop whatever processing it is doing.

Return

- 0 if the effect should continue whatever processing it is doing
- 1 if the effect should abort whatever processing it is doing

OfxStatus (***imageMemoryAlloc**)(*OfxImageEffectHandle* instanceHandle, size_t nBytes, *OfxImageMemoryHandle* *memoryHandle)

Allocate memory from the host's image memory pool.

- `instanceHandle` effect instance to associate with this memory allocation, may be NULL.
- `nBytes` number of bytes to allocate
- `memoryHandle` pointer to the memory handle where a return value is placed

Memory handles allocated by this should be freed by `OfxImageEffectSuiteV1::imageMemoryFree`. To access the memory behind the handle you need to call `OfxImageEffectSuiteV1::imageMemoryLock`.

See `ImageEffectsMemoryAllocation`.

Return

- `kOfxStatOK` if all went well, a valid memory handle is placed in `memoryHandle`
- `kOfxStatErrBadHandle` if `instanceHandle` is not valid, `memoryHandle` is set to NULL
- `kOfxStatErrMemory` if there was not enough memory to satisfy the call, `memoryHandle` is set to NULL

`OfxStatus (*imageMemoryFree)(OfxImageMemoryHandle memoryHandle)`

Frees a memory handle and associated memory.

- `memoryHandle` memory handle returned by `imageMemoryAlloc`

This function frees a memory handle and associated memory that was previously allocated via `OfxImageEffectSuiteV1::imageMemoryAlloc`

If there are outstanding locks, these are ignored and the handle and memory are freed anyway.

See `ImageEffectsMemoryAllocation`.

Return

- `kOfxStatOK` if the memory was cleanly deleted
- `kOfxStatErrBadHandle` if the value of `memoryHandle` was not a valid pointer returned by `OfxImageEffectSuiteV1::imageMemoryAlloc`

`OfxStatus (*imageMemoryLock)(OfxImageMemoryHandle memoryHandle, void **returnedPtr)`

Lock the memory associated with a memory handle and make it available for use.

- `memoryHandle` memory handle returned by `imageMemoryAlloc`
- `returnedPtr` where to the pointer to the locked memory

This function locks the memory associated with a memory handle and returns a pointer to it. The memory will be 16 byte aligned, to allow use of vector operations.

Note that memory locks and unlocks nest.

After the first lock call, the contents of the memory pointer to by `returnedPtr` is undefined. All subsequent calls to lock will return memory with the same contents as the previous call.

Also, if unlocked, then relocked, the memory associated with a memory handle may be at a different address.

See also `OfxImageEffectSuiteV1::imageMemoryUnlock` and `ImageEffectsMemoryAllocation`.

Return

- kOfxStatOK if the memory was locked, a pointer is placed in *returnedPtr*
- kOfxStatErrBadHandle if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*, null is placed in **returnedPtr*
- kOfxStatErrMemory if there was not enough memory to satisfy the call, **returnedPtr* is set to NULL

OfxStatus (***imageMemoryUnlock**)(*OfxImageMemoryHandle* memoryHandle)

Unlock allocated image data.

- *allocatedData* pointer to memory previously returned by *OfxImageEffectSuiteV1::imageAlloc*

This function unlocks a previously locked memory handle. Once completely unlocked, memory associated with a *memoryHandle* is no longer available for use. Attempting to use it results in undefined behaviour.

Note that locks and unlocks nest, and to fully unlock memory you need to match the count of locks placed upon it.

Also note, if you unlock a completely unlocked handle, it has no effect (ie: the lock count can't be negative).

If unlocked, then relocked, the memory associated with a memory handle may be at a different address, however the contents will remain the same.

See also *OfxImageEffectSuiteV1::imageMemoryLock* and *ImageEffectsMemoryAllocation*.

Return

- kOfxStatOK if the memory was unlocked cleanly,
- kOfxStatErrBadHandle if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*, null is placed in **returnedPtr*

Struct OfxInteractSuiteV1

struct **OfxInteractSuiteV1**

OFX suite that allows an effect to interact with an openGL window so as to provide custom interfaces.

Public Members

OfxStatus (***interactSwapBuffers**)(*OfxInteractHandle* interactInstance)

Requests an openGL buffer swap on the interact instance.

OfxStatus (***interactRedraw**)(*OfxInteractHandle* interactInstance)

Requests a redraw of the interact instance.

OfxStatus (***interactGetPropertySet**)(*OfxInteractHandle* interactInstance, *OfxPropertySetHandle* *property)

Gets the property set handle for this interact handle.

Struct `OfxMemorySuiteV1`

struct `OfxMemorySuiteV1`

The OFX suite that implements general purpose memory management.

Use this suite for ordinary memory management functions, where you would normally use `malloc/free` or `new/delete` on ordinary objects.

For images, you should use the memory allocation functions in the image effect suite, as many hosts have specific image memory pools.

Note: C++ plugin developers will need to redefine `new` and `delete` as skins on top of this suite.

Public Members

OfxStatus (***memoryAlloc**)(void *handle, size_t nBytes, void **allocatedData)

Allocate memory.

- `handle` - effect instance to associate with this memory allocation, or `NULL`.
- `nBytes` number of bytes to allocate
- `allocatedData` pointer to the return value. Allocated memory will be aligned for any use.

This function has the host allocate memory using its own memory resources and returns that to the plugin.

Return

- *kOfxStatOK* the memory was successfully allocated
- *kOfxStatErrMemory* the request could not be met and no memory was allocated

OfxStatus (***memoryFree**)(void *allocatedData)

Frees memory.

- `allocatedData` pointer to memory previously returned by *OfxMemorySuiteV1::memoryAlloc*

This function frees any memory that was previously allocated via *OfxMemorySuiteV1::memoryAlloc*.

Return

- *kOfxStatOK* the memory was successfully freed
- *kOfxStatErrBadHandle* `allocatedData` was not a valid pointer returned by *OfxMemorySuiteV1::memoryAlloc*

Struct `OfxMessageSuiteV1`

struct `OfxMessageSuiteV1`

The OFX suite that allows a plug-in to pass messages back to a user. The V2 suite extends on this in a backwards compatible manner.

Public Members

OfxStatus (***message**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a message on the host, using printf style varargs.

- `handle` effect handle (descriptor or instance) the message should be associated with, may be NULL
- `messageType` string describing the kind of message to post, one of the `kOfxMessageType*` constants
- `messageId` plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- `format` printf style format string
- ... printf style varargs list to print

Return

- *kOfxStatOK* - if the message was successfully posted
- *kOfxStatReplyYes* - if the message was of type `kOfxMessageQuestion` and the user reply yes
- *kOfxStatReplyNo* - if the message was of type `kOfxMessageQuestion` and the user reply no
- *kOfxStatFailed* - if the message could not be posted for some reason

Struct `OfxMessageSuiteV2`

struct `OfxMessageSuiteV2`

The OFX suite that allows a plug-in to pass messages back to a user.

This extends *OfxMessageSuiteV1*, and should be considered a replacement to version 1.

Note that this suite has been extended in backwards compatible manner, so that a host can return this struct for both V1 and V2.

Public Members

OfxStatus (***message**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a transient message on the host, using printf style varargs. Same as the V1 message suite call.

- **handle** effect handle (descriptor or instance) the message should be associated with, may be null
- **messageType** string describing the kind of message to post, one of the `kOfxMessageType*` constants
- **messageId** plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- **format** printf style format string
- ... printf style varargs list to print

Return

- *kOfxStatOK* - if the message was successfully posted
- *kOfxStatReplyYes* - if the message was of type `kOfxMessageQuestion` and the user reply yes
- *kOfxStatReplyNo* - if the message was of type `kOfxMessageQuestion` and the user reply no
- *kOfxStatFailed* - if the message could not be posted for some reason

OfxStatus (***setPersistentMessage**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a persistent message on an effect, using printf style varargs, and set error states. New for V2 message suite.

- **handle** effect instance handle the message should be associated with, may NOT be null,
- **messageType** string describing the kind of message to post, should be one of...
 - `kOfxMessageError`
 - `kOfxMessageWarning`
 - `kOfxMessageMessage`
- **messageId** plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- **format** printf style format string
- ... printf style varargs list to print

Persistent messages are associated with an effect handle until explicitly cleared by an effect. So if an error message is posted the error state, and associated message will persist and be displayed on the effect appropriately. (eg: draw a node in red on a node based compositor and display the message when clicked on).

If *messageType* is error or warning, associated error states should be flagged on host applications. Posting an error message implies that the host cannot proceed, a warning allows the host to proceed, whilst a simple message should have no stop anything.

Return

- *kOfxStatOK* - if the message was successfully posted
- *kOfxStatErrBadHandle* - the handle was rubbish
- *kOfxStatFailed* - if the message could not be posted for some reason

OfxStatus (***clearPersistentMessage**)(void *handle)

Clears any persistent message on an effect handle that was set by *OfxMessageSuiteV2::setPersistentMessage*. New for V2 message suite.

- *handle* effect instance handle messages should be cleared from.
- *handle* effect handle (descriptor or instance)

Clearing a message will clear any associated error state.

Return

- *kOfxStatOK* - if the message was successfully cleared
- *kOfxStatErrBadHandle* - the handle was rubbish
- *kOfxStatFailed* - if the message could not be cleared for some reason

Struct OfxMultiThreadSuiteV1

struct **OfxMultiThreadSuiteV1**

OFX suite that provides simple SMP style multi-processing.

Public Members

OfxStatus (***multiThread**)(OfxThreadFunctionV1 func, unsigned int nThreads, void *customArg)

Function to spawn SMP threads.

- *func* function to call in each thread.
- *nThreads* number of threads to launch
- *customArg* parameter to pass to *customArg* of *func* in each thread.

This function will spawn *nThreads* separate threads of computation (typically one per CPU) to allow something to perform symmetric multi processing. Each thread will call 'func' passing in the index of the thread and the number of threads actually launched.

multiThread will not return until all the spawned threads have returned. It is up to the host how it waits for all the threads to return (busy wait, blocking, whatever).

nThreads can be more than the value returned by *multiThreadNumCPUs*, however the threads will be limited to the number of CPUs returned by *multiThreadNumCPUs*.

This function cannot be called recursively.

Return

- *kOfxStatOK*, the function func has executed and returned successfully
- *kOfxStatFailed*, the threading function failed to launch
- *kOfxStatErrExists*, failed in an attempt to call multiThread recursively,

OfxStatus (***multiThreadNumCPUs**)(unsigned int *nCPUs)

Function which indicates the number of CPUs available for SMP processing.

- nCPUs pointer to an integer where the result is returned

This value may be less than the actual number of CPUs on a machine, as the host may reserve other CPUs for itself.

Return

- *kOfxStatOK*, all was OK and the maximum number of threads is in nThreads.
- *kOfxStatFailed*, the function failed to get the number of CPUs

OfxStatus (***multiThreadIndex**)(unsigned int *threadIndex)

Function which indicates the index of the current thread.

- threadIndex pointer to an integer where the result is returned

This function returns the thread index, which is the same as the *threadIndex* argument passed to the *Ofx-ThreadFunctionV1*.

If there are no threads currently spawned, then this function will set threadIndex to 0

Return

- *kOfxStatOK*, all was OK and the maximum number of threads is in nThreads.
- *kOfxStatFailed*, the function failed to return an index

int (***multiThreadIsSpawnedThread**)(void)

Function to enquire if the calling thread was spawned by multiThread.

Return

- 0 if the thread is not one spawned by multiThread
- 1 if the thread was spawned by multiThread

OfxStatus (***mutexCreate**)(*OfxMutexHandle* *mutex, int lockCount)

Create a mutex.

- mutex where the new handle is returned
- count initial lock count on the mutex. This can be negative.

Creates a new mutex with lockCount locks on the mutex intially set.

Return

- kOfxStatOK - mutex is now valid and ready to go

OfxStatus (***mutexDestroy**)(const *OfxMutexHandle* mutex)

Destroy a mutex.

Destroys a mutex initially created by `mutexCreate`.

Return

- `kOfxStatOK` - if it destroyed the mutex
- `kOfxStatErrBadHandle` - if the handle was bad

OfxStatus (***mutexLock**)(const *OfxMutexHandle* mutex)

Blocking lock on the mutex.

This tries to lock a mutex and blocks the thread it is in until the lock succeeds.

A successful lock causes the mutex's lock count to be increased by one and to block any other calls to lock the mutex until it is unlocked.

Return

- `kOfxStatOK` - if it got the lock
- `kOfxStatErrBadHandle` - if the handle was bad

OfxStatus (***mutexUnlock**)(const *OfxMutexHandle* mutex)

Unlock the mutex.

This unlocks a mutex. Unlocking a mutex decreases its lock count by one.

Return

- `kOfxStatOK` if it released the lock
- `kOfxStatErrBadHandle` if the handle was bad

OfxStatus (***mutexTryLock**)(const *OfxMutexHandle* mutex)

Non blocking attempt to lock the mutex.

This attempts to lock a mutex, if it cannot, it returns and says so, rather than blocking.

A successful lock causes the mutex's lock count to be increased by one, if the lock did not succeed, the call returns immediately and the lock count remains unchanged.

Return

- `kOfxStatOK` - if it got the lock
- `kOfxStatFailed` - if it did not get the lock
- `kOfxStatErrBadHandle` - if the handle was bad

Struct `OfxOpenCLProgramSuiteV1`

struct `OfxOpenCLProgramSuiteV1`

OFX suite that allows a plug-in to get OpenCL programs compiled.

This is an optional suite the host can provide for building OpenCL programs for the plug-in, as an alternative to calling `clCreateProgramWithSource / clBuildProgram`. There are two advantages to doing this: The host can add flags (such as `-cl-denorms-are-zero`) to the build call, and may also cache program binaries for performance (however, if the source of the program or the OpenCL environment changes, the host must recompile so some mechanism such as hashing must be used).

Public Members

OfxStatus (***compileProgram**)(const char *pszProgramSource, int fOptional, void *pResult)

Compiles the OpenCL program.

Struct `OfxParameterSuiteV1`

struct `OfxParameterSuiteV1`

The OFX suite used to define and manipulate user visible parameters.

Keyframe Handling

These functions allow the plug-in to delete and get information about keyframes.

To set keyframes, use *paramSetValueAtTime()*.

`paramGetKeyTime` and `paramGetKeyIndex` use indices to refer to keyframes. Keyframes are stored by the host in increasing time order, so `time(kf[i]) < time(kf[i+1])`. Keyframe indices will change whenever keyframes are added, deleted, or moved in time, whether by the host or by the plug-in. They may vary between actions if the user changes a keyframe. The keyframe indices will not change within a single action.

OfxStatus (***paramGetNumKeys**)(*OfxParamHandle* paramHandle, unsigned int *numberOfKeys)

Returns the number of keyframes in the parameter.

- `paramHandle` parameter handle to interrogate
- `numberOfKeys` pointer to integer where the return value is placed

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Returns the number of keyframes in the parameter.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetKeyTime**)(*OfxParamHandle* paramHandle, unsigned int nthKey, *OfxTime* *time)

Returns the time of the nth key.

- **paramHandle** parameter handle to interrogate
- **nthKey** which key to ask about (0 to paramGetNumKeys -1), ordered by time
- **time** pointer to *OfxTime* where the return value is placed

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrBadIndex* - the nthKey does not exist

OfxStatus (***paramGetKeyIndex**)(*OfxParamHandle* paramHandle, *OfxTime* time, int direction, int *index)

Finds the index of a keyframe at/before/after a specified time.

- **paramHandle** parameter handle to search
- **time** what time to search from
- **direction**
 - == 0 indicates search for a key at the indicated time (some small delta)
 - > 0 indicates search for the next key after the indicated time
 - < 0 indicates search for the previous key before the indicated time
- **index** pointer to an integer which in which the index is returned set to -1 if no key was found

Return

- *kOfxStatOK* - all was OK
- *kOfxStatFailed* - if the search failed to find a key
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramDeleteKey**)(*OfxParamHandle* paramHandle, *OfxTime* time)

Deletes a keyframe if one exists at the given time.

- **paramHandle** parameter handle to delete the key from
- **time** time at which a keyframe is

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrBadIndex* - no key at the given time

OfxStatus (***paramDeleteAllKeys**)(*OfxParamHandle* paramHandle)

Deletes all keyframes from a parameter.

- paramHandle parameter handle to delete the keys from
- name parameter to delete the keyframes from is

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

Public Members

OfxStatus (***paramDefine**)(*OfxParamSetHandle* paramSet, const char *paramType, const char *name, *OfxPropertySetHandle* *propertySet)

Defines a new parameter of the given type in a describe action.

- paramSet handle to the parameter set descriptor that will hold this parameter
- paramType type of the parameter to create, one of the *kOfxParamType** #defines
- name unique name of the parameter
- propertySet if not null, a pointer to the parameter descriptor's property set will be placed here.

This function defines a parameter in a parameter set and returns a property set which is used to describe that parameter.

This function does not actually create a parameter, it only says that one should exist in any subsequent instances. To fetch an parameter instance *paramGetHandle* must be called on an instance.

This function can always be called in one of a plug-in's "describe" functions which defines the parameter sets common to all instances of a plugin.

Return

- *kOfxStatOK* - the parameter was created correctly
- *kOfxStatErrBadHandle* - if the plugin handle was invalid
- *kOfxStatErrExists* - if a parameter of that name exists already in this plugin
- *kOfxStatErrUnknown* - if the type is unknown
- *kOfxStatErrUnsupported* - if the type is known but unsupported

OfxStatus (***paramGetHandle**)(*OfxParamSetHandle* paramSet, const char *name, *OfxParamHandle* *param, *OfxPropertySetHandle* *propertySet)

Retrieves the handle for a parameter in a given parameter set.

- paramSet instance of the plug-in to fetch the property handle from

- name parameter to ask about
- param pointer to a param handle, the value is returned here
- propertySet if not null, a pointer to the parameter's property set will be placed here.

Parameter handles retrieved from an instance are always distinct in each instance. The paramter handle is valid for the life-time of the instance. Parameter handles in instances are distinct from paramter handles in plugins. You cannot call this in a plugin's describe function, as it needs an instance to work on.

Return

- *kOfxStatOK* - the parameter was found and returned
- *kOfxStatErrBadHandle* - if the plugin handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramSetGetPropertySet**)(*OfxParamSetHandle* paramSet, *OfxPropertySetHandle* *propHandle)

Retrieves the property set handle for the given parameter set.

- paramSet parameter set to get the property set for
- propHandle pointer to a the property set handle, value is returned here

Note: The property handle belonging to a parameter set is the same as the property handle belonging to the plugin instance.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramGetPropertySet**)(*OfxParamHandle* param, *OfxPropertySetHandle* *propHandle)

Retrieves the property set handle for the given parameter.

- param parameter to get the property set for
- propHandle pointer to a the property set handle, value is returned here

The property handle is valid for the lifetime of the parameter, which is the lifetime of the instance that owns the parameter

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramGetValue**)(*OfxParamHandle* paramHandle, ...)

Gets the current value of a parameter,.

- `paramHandle` parameter handle to fetch value from
- ... one or more pointers to variables of the relevant type to hold the parameter's value

This gets the current value of a parameter. The `varargs ...` argument needs to be pointer to C variables of the relevant type for this parameter. Note that params with multiple values (eg `Colour`) take multiple args here. For example...

```
OfxParamHandle myDoubleParam, *myColourParam;
ofxHost->paramGetHandle(instance, "myDoubleParam", &myDoubleParam);
double myDoubleValue;
ofxHost->paramGetValue(myDoubleParam, &myDoubleValue);
ofxHost->paramGetHandle(instance, "myColourParam", &myColourParam);
double myR, myG, myB;
ofxHost->paramGetValue(myColourParam, &myR, &myG, &myB);
```

Note: `paramGetValue` should only be called from within a *kOfxActionInstanceChanged* or interact action and never from the render actions (which should always use `paramGetValueAtTime`).

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetValueAtTime**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Gets the value of a parameter at a specific time.

- `paramHandle` parameter handle to fetch value from
- `time` at what point in time to look up the parameter
- ... one or more pointers to variables of the relevant type to hold the parameter's value

This gets the current value of a parameter. The `varargs` needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteV1::paramGetValue* for notes on the `varargs` list

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetDerivative**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Gets the derivative of a parameter at a specific time.

- `paramHandle` parameter handle to fetch value from
- `time` at what point in time to look up the parameter

- ... one or more pointers to variables of the relevant type to hold the parameter's derivative

This gets the derivative of the parameter at the indicated time.

The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteV1::paramGetValue* for notes on the varargs list.

Only double and colour params can have their derivatives found.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetIntegral**)(*OfxParamHandle* paramHandle, *OfxTime* time1, *OfxTime* time2, ...)

Gets the integral of a parameter over a specific time range,.

- paramHandle parameter handle to fetch integral from
- time1 where to start evaluating the integral
- time2 where to stop evaluating the integral
- ... one or more pointers to variables of the relevant type to hold the parameter's integral

This gets the integral of the parameter over the specified time range.

The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteV1::paramGetValue* for notes on the varargs list.

Only double and colour params can be integrated.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramSetValue**)(*OfxParamHandle* paramHandle, ...)

Sets the current value of a parameter.

- paramHandle parameter handle to set value in
- ... one or more variables of the relevant type to hold the parameter's value

This sets the current value of a parameter. The varargs ... argument needs to be values of the relevant type for this parameter. Note that params with multiple values (eg Colour) take multiple args here. For example...

```
ofxHost->paramSetValue(instance, "myDoubleParam", double(10));
ofxHost->paramSetValue(instance, "myColourParam", double(pix.r), double(pix.
g), double(pix.b));
```

Note: paramSetValue should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramSetValueAtTime**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Keyframes the value of a parameter at a specific time.

- **paramHandle** parameter handle to set value in
- **time** at what point in time to set the keyframe
- ... one or more variables of the relevant type to hold the parameter's value

This sets a keyframe in the parameter at the indicated time to have the indicated value. The varargs ... argument needs to be values of the relevant type for this parameter. See the note on *OfxParameterSuiteV1::paramSetValue* for more detail

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Note: **paramSetValueAtTime** should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramCopy**)(*OfxParamHandle* paramTo, *OfxParamHandle* paramFrom, *OfxTime* dstOffset, const *OfxRangeD* *frameRange)

Copies one parameter to another, including any animation etc...

- **paramTo** parameter to set
- **paramFrom** parameter to copy from
- **dstOffset** temporal offset to apply to keys when writing to the paramTo
- **frameRange** if paramFrom has animation, and frameRange is not null, only this range of keys will be copied

This copies the value of *paramFrom* to *paramTo*, including any animation it may have. All the previous values in *paramTo* will be lost.

To choose all animation in *paramFrom* set *frameRange* to [0, 0]

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Pre

- Both parameters must be of the same type.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramEditBegin**)(*OfxParamSetHandle* paramSet, const char *name)

Used to group any parameter changes for undo/redo purposes.

- paramSet the parameter set in which this is happening
- name label to attach to any undo/redo string UTF8

If a plugin calls paramSetValue/paramSetValueAtTime on one or more parameters, either from custom GUI interaction or some analysis of imagery etc.. this is used to indicate the start of a set of a parameter changes that should be considered part of a single undo/redo block.

See also *OfxParameterSuiteV1::paramEditEnd*

Note: paramEditBegin should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the instance handle was invalid

OfxStatus (***paramEditEnd**)(*OfxParamSetHandle* paramSet)

Used to group any parameter changes for undo/redo purposes.

- paramSet parameter set in which this is happening

If a plugin calls paramSetValue/paramSetValueAtTime on one or more parameters, either from custom GUI interaction or some analysis of imagery etc.. this is used to indicate the end of a set of parameter changes that should be considered part of a single undo/redo block

See also *OfxParameterSuiteV1::paramEditBegin*

Note: paramEditEnd should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the instance handle was invalid

Struct `OfxParametricParameterSuiteV1`

struct **OfxParametricParameterSuiteV1**

The OFX suite used to define and manipulate ‘parametric’ parameters.

This is an optional suite.

Parametric parameters are in effect ‘functions’ a plug-in can ask a host to arbitrarily evaluate for some value ‘x’. A classic use case would be for constructing look-up tables, a plug-in would ask the host to evaluate one at multiple values from 0 to 1 and use that to fill an array.

A host would probably represent this to a user as a cubic curve in a standard curve editor interface, or possibly through scripting. The user would then use this to define the ‘shape’ of the parameter.

The evaluation of such params is not the same as animation, they are returning values based on some arbitrary argument orthogonal to time, so to evaluate such a param, you need to pass a parametric position and time.

Often, you would want such a parametric parameter to be multi-dimensional, for example, a colour look-up table might want three values, one for red, green and blue. Rather than declare three separate parametric parameters, it would be better to have one such parameter with multiple values in it.

The major complication with these parameters is how to allow a plug-in to set values, and defaults. The default default value of a parametric curve is to be an identity lookup. If a plugin wishes to set a different default value for a curve, it can use the suite to set key/value pairs on the *descriptor* of the param. When a new instance is made, it will have these curve values as a default.

Public Members

OfxStatus (***parametricParamGetValue**)(*OfxParamHandle* param, int curveIndex, *OfxTime* time, double parametricPosition, double *returnValue)

Evaluates a parametric parameter.

- param handle to the parametric parameter
- curveIndex which dimension to evaluate
- time the time to evaluate to the parametric param at
- parametricPosition the position to evaluate the parametric param at
- returnValue pointer to a double where a value is returned

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrBadIndex* - the curve index was invalid

OfxStatus (***parametricParamGetNControlPoints**)(*OfxParamHandle* param, int curveIndex, double time, int *returnValue)

Returns the number of control points in the parametric param.

- param handle to the parametric parameter

- `curveIndex` which dimension to check
- `time` the time to check
- `returnValue` pointer to an integer where the value is returned.

Return

- `kOfxStatOK` - all was fine
- `kOfxStatErrBadHandle` - if the paramter handle was invalid
- `kOfxStatErrBadIndex` - the curve index was invalid

OfxStatus (***parametricParamGetNthControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, int nthCtl, double *key, double *value)

Returns the key/value pair of the nth control point.

- param handle to the parametric parameter
- `curveIndex` which dimension to check
- `time` the time to check
- `nthCtl` the nth control point to get the value of
- key pointer to a double where the key will be returned
- value pointer to a double where the value will be returned

Return

- `kOfxStatOK` - all was fine
- `kOfxStatErrBadHandle` - if the paramter handle was invalid
- `kOfxStatErrUnknown` - if the type is unknown

OfxStatus (***parametricParamSetNthControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, int nthCtl, double key, double value, bool addAnimationKey)

Modifies an existing control point on a curve.

- param handle to the parametric parameter
- `curveIndex` which dimension to set
- `time` the time to set the value at
- `nthCtl` the control point to modify
- key key of the control point
- value value of the control point
- `addAnimationKey` if the param is an animatable, setting this to true will force an animation keyframe to be set as well as a curve key, otherwise if false, a key will only be added if the curve is already animating.

This modifies an existing control point. Note that by changing key, the order of the control point may be modified (as you may move it before or after another point). So be careful when iterating over a curve's control points and you change a key.

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***parametricParamAddControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, double key, double value, bool addAnimationKey)

Adds a control point to the curve.

- param handle to the parametric parameter
- curveIndex which dimension to set
- time the time to set the value at
- key key of the control point
- value value of the control point
- addAnimationKey if the param is an animatable, setting this to true will force an animation keyframe to be set as well as a curve key, otherwise if false, a key will only be added if the curve is already animating.

This will add a new control point to the given dimension of a parametric parameter. If a key exists sufficiently close to 'key', then it will be set to the indicated control point.

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***parametricParamDeleteControlPoint**)(*OfxParamHandle* param, int curveIndex, int nthCtl)

Deletes the nth control point from a parametric param.

- param handle to the parametric parameter
- curveIndex which dimension to delete
- nthCtl the control point to delete

OfxStatus (***parametricParamDeleteAllControlPoints**)(*OfxParamHandle* param, int curveIndex)

Delete all curve control points on the given param.

- param handle to the parametric parameter
- curveIndex which dimension to clear

Struct OfxPlugin

struct **OfxPlugin**

The structure that defines a plug-in to a host.

This structure is the first element in any plug-in structure using the OFX plug-in architecture. By examining its members a host can determine the API that the plug-in implements, the version of that API, its name and version.

For details see Architecture.

Public Members

const char ***pluginApi**

Defines the type of the plug-in, this will tell the host what the plug-in does. e.g.: an image effects plug-in would be a “OfxImageEffectPlugin”

int **apiVersion**

Defines the version of the pluginApi that this plug-in implements

const char ***pluginIdentifier**

String that uniquely labels the plug-in among all plug-ins that implement an API. It need not necessarily be human sensible, however the preference is to use reverse internet domain name of the developer, followed by a ‘.’ then by a name that represents the plug-in.. It must be a legal ASCII string and have no whitespace in the name and no non printing chars. For example “uk.co.somesoftwarehouse.myPlugin”

unsigned int **pluginVersionMajor**

Major version of this plug-in, this gets incremented when backwards compatibility is broken.

unsigned int **pluginVersionMinor**

Major version of this plug-in, this gets incremented when software is changed, but does not break backwards compatibility.

void (***setHost**)(OfxHost *host)

Function the host uses to connect the plug-in to the host’s api fetcher.

- **fetchApi** pointer to host’s API fetcher

Mandatory function.

The very first function called in a plug-in. The plug-in *must not* call any OFX functions within this, it must only set its local copy of the host pointer.

It is recommended that hosts should return the same host and suite pointers to all plugins in the same shared lib or bundle.

Pre

- nothing else has been called

Post

- the pointer suite is valid until the plug-in is unloaded

OfxPluginEntryPoint ***mainEntry**

Main entry point for plug-ins.

Mandatory function.

The exact set of actions is determined by the plug-in API that is being implemented, however all plug-ins can perform several actions. For the list of actions consult OFX Actions.

Preconditions

- setHost has been called

Struct OfxPointD

struct **OfxPointD**

Defines two dimensional double point.

Public Members

double **x**

double **y**

Struct OfxPointI

struct **OfxPointI**

Defines two dimensional integer point.

Public Members

int **x**

int **y**

Struct OfxProgressSuiteV1

struct **OfxProgressSuiteV1**

A suite that provides progress feedback from a plugin to an application.

A plugin instance can initiate, update and close a progress indicator with this suite.

This is an optional suite in the Image Effect API.

API V1.4: Amends the documentation of progress suite V1 so that it is expected that it can be raised in a modal manner and have a “cancel” button when invoked in instanceChanged. Plugins that perform analysis post an

appropriate message, raise the progress monitor in a modal manner and should poll to see if processing has been aborted. Any cancellation should be handled gracefully by the plugin (eg: reset analysis parameters to default values), clear allocated memory...

Many hosts already operate as described above. `kOfxStatReplyNo` should be returned to the plugin during `progressUpdate` when the user presses cancel.

Suite V2: Adds an ID that can be looked up for internationalisation and so on. When a new version is introduced, because plug-ins need to support old versions, and plug-in's new releases are not necessary in synch with hosts (or users don't immediately update), best practice is to support the 2 suite versions. That is, the plugin should check if V2 exists; if not then check if V1 exists. This way a graceful transition is guaranteed. So plugin should `fetchSuite` passing 2, (`OfxProgressSuiteV2*`) `fetchSuite(mHost->mHost->host, kOfxProgressSuite,2)`; and if no success pass (`OfxProgressSuiteV1*`) `fetchSuite(mHost->mHost->host, kOfxProgressSuite,1)`;

Public Members

OfxStatus (***progressStart**)(void *effectInstance, const char *label)

Initiate a progress bar display.

Call this to initiate the display of a progress bar.

- `effectInstance` the instance of the plugin this progress bar is associated with. It cannot be NULL.
- `label` a text label to display in any message portion of the progress object's user interface. A UTF8 string.

Pre

- There is no currently ongoing progress display for this instance.

Return

- `kOfxStatOK` - the handle is now valid for use
- `kOfxStatFailed` - the progress object failed for some reason
- `kOfxStatErrBadHandle` - `effectInstance` was invalid

OfxStatus (***progressUpdate**)(void *effectInstance, double progress)

Indicate how much of the processing task has been completed and reports on any abort status.

- `effectInstance` the instance of the plugin this progress bar is associated with. It cannot be NULL.
- `progress` a number between 0.0 and 1.0 indicating what proportion of the current task has been processed.

Return

- `kOfxStatOK` - the progress object was successfully updated and the task should continue
- `kOfxStatReplyNo` - the progress object was successfully updated and the task should abort
- `kOfxStatErrBadHandle` - the progress handle was invalid,

OfxStatus (***progressEnd**)(void *effectInstance)

Signal that we are finished with the progress meter.

Call this when you are done with the progress meter and no longer need it displayed.

- **effectInstance** the instance of the plugin this progress bar is associated with. It cannot be NULL.

Post

- you can no longer call `progressUpdate` on the instance

Return

- *kOfxStatOK* - the progress object was successfully closed
- *kOfxStatErrBadHandle* - the progress handle was invalid,

Struct `OfxProgressSuiteV2`

struct `OfxProgressSuiteV2`

Public Members

OfxStatus (***progressStart**)(void *effectInstance, const char *message, const char *messageid)

Initiate a progress bar display.

Call this to initiate the display of a progress bar.

- **effectInstance** the instance of the plugin this progress bar is associated with. It cannot be NULL.
- **message** a text label to display in any message portion of the progress object's user interface. A UTF8 string.
- **messageId** plugin-specified id to associate with this message. If overriding the message in an XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur. New in V2 of this suite.

Pre

- There is no currently ongoing progress display for this instance.

Return

- *kOfxStatOK* - the handle is now valid for use
- *kOfxStatFailed* - the progress object failed for some reason
- *kOfxStatErrBadHandle* - effectInstance was invalid

OfxStatus (***progressUpdate**)(void *effectInstance, double progress)

Indicate how much of the processing task has been completed and reports on any abort status.

- **effectInstance** the instance of the plugin this progress bar is associated with. It cannot be NULL.

- **progress** a number between 0.0 and 1.0 indicating what proportion of the current task has been processed.

Return

- *kOfxStatOK* - the progress object was successfully updated and the task should continue
- *kOfxStatReplyNo* - the progress object was successfully updated and the task should abort
- *kOfxStatErrBadHandle* - the progress handle was invalid,

OfxStatus (***progressEnd**)(void *effectInstance)

Signal that we are finished with the progress meter.

Call this when you are done with the progress meter and no longer need it displayed.

- **effectInstance** the instance of the plugin this progress bar is associated with. It cannot be NULL.

Post

- you can no longer call `progressUpdate` on the instance

Return

- *kOfxStatOK* - the progress object was successfully closed
- *kOfxStatErrBadHandle* - the progress handle was invalid,

Struct `OfxPropertySuiteV1`

struct **OfxPropertySuiteV1**

The OFX suite used to access properties on OFX objects.

Public Members

OfxStatus (***propSetPointer**)(*OfxPropertySetHandle* properties, const char *property, int index, void *value)

Set a single value in a pointer property.

- **properties** handle of the thing holding the property
- **property** string labelling the property
- **index** for multidimensional properties and is dimension of the one we are setting
- **value** value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

- *kOfxStatErrValue*

OfxStatus (***propSetString**)(*OfxPropertySetHandle* properties, const char *property, int index, const char *value)

Set a single value in a string property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *index* for multidimensional properties and is dimension of the one we are setting
- *value* value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetDouble**)(*OfxPropertySetHandle* properties, const char *property, int index, double value)

Set a single value in a double property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *index* for multidimensional properties and is dimension of the one we are setting
- *value* value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetInt**)(*OfxPropertySetHandle* properties, const char *property, int index, int value)

Set a single value in an int property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *index* for multidimensional properties and is dimension of the one we are setting

- value value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetPointerN**)(*OfxPropertySetHandle* properties, const char *property, int count, void *const *value)

Set multiple values of the pointer property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *count* number of values we are setting in that property (ie: indicies 0..count-1)
- *value* pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetStringN**)(*OfxPropertySetHandle* properties, const char *property, int count, const char *const *value)

Set multiple values of a string property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *count* number of values we are setting in that property (ie: indicies 0..count-1)
- *value* pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetDoubleN**)(*OfxPropertySetHandle* properties, const char *property, int count, const double *value)

Set multiple values of a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are setting in that property (ie: indices 0..count-1)
- `value` pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetIntN**)(*OfxPropertySetHandle* properties, const char *property, int count, const int *value)

Set multiple values of an int property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are setting in that property (ie: indices 0..count-1)
- `value` pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propGetPointer**)(*OfxPropertySetHandle* properties, const char *property, int index, void **value)

Get a single value from a pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetString**)(*OfxPropertySetHandle* properties, const char *property, int index, char **value)

Get a single value of a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetDouble**)(*OfxPropertySetHandle* properties, const char *property, int index, double *value)

Get a single value of a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

See the note `ArchitectureStrings` for how to deal with strings.

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetInt**)(*OfxPropertySetHandle* properties, const char *property, int index, int *value)

Get a single value of an int property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- *`kOfxStatOK`*
- *`kOfxStatErrBadHandle`*
- *`kOfxStatErrUnknown`*
- *`kOfxStatErrBadIndex`*

`OfxStatus` (***propGetPointerN**)(*`OfxPropertySetHandle`* `properties`, `const char *property`, `int count`, `void **value`)

Get multiple values of a pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indices 0..count-1)
- `value` pointer to an array of where we will return the property values

Return

- *`kOfxStatOK`*
- *`kOfxStatErrBadHandle`*
- *`kOfxStatErrUnknown`*
- *`kOfxStatErrBadIndex`*

`OfxStatus` (***propGetStringN**)(*`OfxPropertySetHandle`* `properties`, `const char *property`, `int count`, `char **value`)

Get multiple values of a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indices 0..count-1)
- `value` pointer to an array of where we will return the property values

See the note `ArchitectureStrings` for how to deal with strings.

Return

- *`kOfxStatOK`*
- *`kOfxStatErrBadHandle`*
- *`kOfxStatErrUnknown`*

- *kOfxStatErrBadIndex*

OfxStatus (***propGetDoubleN**)(*OfxPropertySetHandle* properties, const char *property, int count, double *value)

Get multiple values of a double property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *count* number of values we are getting of that property (ie: indices 0..count-1)
- *value* pointer to an array of where we will return the property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetIntN**)(*OfxPropertySetHandle* properties, const char *property, int count, int *value)

Get multiple values of an int property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *count* number of values we are getting of that property (ie: indices 0..count-1)
- *value* pointer to an array of where we will return the property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propReset**)(*OfxPropertySetHandle* properties, const char *property)

Resets all dimensions of a property to its default value.

- *properties* handle of the thing holding the property
- *property* string labelling the property we are resetting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*

- *kOfxStatErrUnknown*

OfxStatus (***propGetDimension**)(*OfxPropertySetHandle* properties, const char *property, int *count)

Gets the dimension of the property.

- properties handle of the thing holding the property
- property string labelling the property we are resetting
- count pointer to an integer where the value is returned

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*

Struct OfxRGBAColourB

struct **OfxRGBAColourB**

Defines an 8 bit per component RGBA pixel.

Public Members

unsigned char **r**

unsigned char **g**

unsigned char **b**

unsigned char **a**

Struct OfxRGBAColourD

struct **OfxRGBAColourD**

Defines a double precision floating point component RGBA pixel.

Public Membersdouble **r**double **g**double **b**double **a****Struct OfxRGBAColourF**struct **OfxRGBAColourF**

Defines a floating point component RGBA pixel.

Public Membersfloat **r**float **g**float **b**float **a****Struct OfxRGBAColourS**struct **OfxRGBAColourS**

Defines a 16 bit per component RGBA pixel.

Public Membersunsigned short **r**unsigned short **g**unsigned short **b**unsigned short **a**

Struct `OfxRGBColourB`

struct `OfxRGBColourB`

Defines an 8 bit per component RGB pixel.

Public Members

unsigned char `r`

unsigned char `g`

unsigned char `b`

Struct `OfxRGBColourD`

struct `OfxRGBColourD`

Defines a double precision floating point component RGB pixel.

Public Members

double `r`

double `g`

double `b`

Struct `OfxRGBColourF`

struct `OfxRGBColourF`

Defines a floating point component RGB pixel.

Public Members

float `r`

float `g`

float `b`

Struct OfxRGBColourS

struct **OfxRGBColourS**

Defines a 16 bit per component RGB pixel.

Public Members

unsigned short **r**

unsigned short **g**

unsigned short **b**

Struct OfxRanged

struct **OfxRanged**

Defines one dimensional double bounds.

Public Members

double **min**

double **max**

Struct OfxRangel

struct **OfxRangeI**

Defines one dimensional integer bounds.

Public Members

int **min**

int **max**

Struct `OfxRectD`

struct **OfxRectD**

Defines two dimensional double region.

Regions are $x1 \leq x < x2$

Infinite regions are flagged by setting

- $x1 = kOfxFlagInfiniteMin$
- $y1 = kOfxFlagInfiniteMin$
- $x2 = kOfxFlagInfiniteMax$
- $y2 = kOfxFlagInfiniteMax$

Public Members

double **x1**

double **y1**

double **x2**

double **y2**

Struct `OfxRectI`

struct **OfxRectI**

Defines two dimensional integer region.

Regions are $x1 \leq x < x2$

Infinite regions are flagged by setting

- $x1 = kOfxFlagInfiniteMin$
- $y1 = kOfxFlagInfiniteMin$
- $x2 = kOfxFlagInfiniteMax$
- $y2 = kOfxFlagInfiniteMax$

Public Members

int **x1**

int **y1**

int **x2**

int **y2**

Struct `OfxTimeLineSuiteV1`

struct **OfxTimeLineSuiteV1**

Suite to control timelines.

This suite is used to enquire and control a timeline associated with a plug-in instance.

This is an optional suite in the Image Effect API.

Public Members

OfxStatus (***getTime**)(void *instance, double *time)

Get the time value of the timeline that is controlling to the indicated effect.

- `instance` is the instance of the effect changing the timeline, cast to a void *
- `time` pointer through which the timeline value should be returned

This function returns the current time value of the timeline associated with the effect instance.

Return

- *kOfxStatOK* - the time enquiry was successful
- *kOfxStatFailed* - the enquiry failed for some host specific reason
- *kOfxStatErrBadHandle* - the effect handle was invalid

OfxStatus (***gotoTime**)(void *instance, double time)

Move the timeline control to the indicated time.

- `instance` is the instance of the effect changing the timeline, cast to a void *
- `time` is the time to change the timeline to. This is in the temporal coordinate system of the effect.

This function moves the timeline to the indicated frame and returns. Any side effects of the timeline change are also triggered and completed before this returns (for example instance changed actions and renders if the output of the effect is being viewed).

Return

- *kOfxStatOK* - the time was changed successfully, will all side effects if the change completed
- *kOfxStatFailed* - the change failed for some host specific reason
- *kOfxStatErrBadHandle* - the effect handle was invalid
- *kOfxStatErrValue* - the time was an illegal value

OfxStatus (***getTimeBounds**)(void *instance, double *firstTime, double *lastTime)

Get the current bounds on a timeline.

- *instance* is the instance of the effect changing the timeline, cast to a void *
- *firstTime* is the first time on the timeline. This is in the temporal coordinate system of the effect.
- *lastTime* is last time on the timeline. This is in the temporal coordinate system of the effect.

This function

Return

- *kOfxStatOK* - the time enquiry was successful
- *kOfxStatFailed* - the enquiry failed for some host specific reason
- *kOfxStatErrBadHandle* - the effect handle was invalid

Struct OfxYUVAColourB

struct **OfxYUVAColourB**

Defines an 8 bit per component YUVA pixel — *ofxPixels.h* Deprecated in 1.3, removed in 1.4.

Public Members

unsigned char **y**

unsigned char **u**

unsigned char **v**

unsigned char **a**

Struct OfxYUVAColourF

struct **OfxYUVAColourF**

Defines an floating point component YUVA pixel — *ofxPixels.h*.

Deprecated:

- Deprecated in 1.3, removed in 1.4

Public Membersfloat **y**float **u**float **v**float **a****Struct OfxYUVAColourS**struct **OfxYUVAColourS**Defines an 16 bit per component YUVA pixel — *ofxPixels.h*.*Deprecated:*

- Deprecated in 1.3, removed in 1.4

Public Membersunsigned short **y**unsigned short **u**unsigned short **v**unsigned short **a**

1.22 Complete Reference Index

struct **OfxDialogSuiteV1***#include* <ofxDialog.h>

Public Members

OfxStatus (***RequestDialog**)(void *user_data)

Request the host to send a `kOfxActionDialog` to the plugin from its UI thread.

Pre

- `user_data`: A pointer to any user data

Post

Return

- *kOfxStatOK* - The host has queued the request and will send an 'OfxActionDialog'
- *kOfxStatFailed* - The host has no provision for this or can not deal with it currently.

OfxStatus (***NotifyRedrawPending**)(void)

Inform the host of redraw event so it can redraw itself. If the host runs fullscreen in OpenGL, it would otherwise not receive redraw event when a dialog in front would catch all events.

Pre

Post

Return

- *kOfxStatReplyDefault*

struct **OfxDrawSuiteV1**

#include <ofxDrawSuite.h> OFX suite that allows an effect to draw to a host-defined display context.

Public Members

OfxStatus (***getColour**)(*OfxDrawContextHandle* context, *OfxStandardColour* std_colour, *OfxRGBAColourF* *colour)

Retrieves the host's desired draw colour for.

- `context` draw context
- `std_colour` desired colour type
- `colour` returned RGBA colour

Return

- *kOfxStatOK* - the colour was returned
- *kOfxStatErrValue* - `std_colour` was invalid
- *kOfxStatFailed* - failure, e.g. if function is called outside `kOfxInteractActionDraw`

OfxStatus (***setColour**)(*OfxDrawContextHandle* context, const *OfxRGBAColourF* *colour)

Sets the colour for future drawing operations (lines, filled shapes and text)

- context draw context
- colour RGBA colour

The host should use “over” compositing when using a non-opaque colour.

Return

- *kOfxStatOK* - the colour was changed
- *kOfxStatFailed* - failure, e.g. if function is called outside `kOfxInteractActionDraw`

OfxStatus (***setLineWidth**)(*OfxDrawContextHandle* context, float width)

Sets the line width for future line drawing operations.

- context draw context
- width line width

Use width 0 for a single pixel line or non-zero for a smooth line of the desired width

The host should adjust for screen density.

Return

- *kOfxStatOK* - the width was changed
- *kOfxStatFailed* - failure, e.g. if function is called outside `kOfxInteractActionDraw`

OfxStatus (***setLineStipple**)(*OfxDrawContextHandle* context, *OfxDrawLineStipplePattern* pattern)

Sets the stipple pattern for future line drawing operations.

- context draw context
- pattern desired stipple pattern

Return

- *kOfxStatOK* - the pattern was changed
- *kOfxStatErrValue* - pattern was not valid
- *kOfxStatFailed* - failure, e.g. if function is called outside `kOfxInteractActionDraw`

OfxStatus (***draw**)(*OfxDrawContextHandle* context, *OfxDrawPrimitive* primitive, const *OfxPointD* *points, int point_count)

Draws a primitive of the desired type.

- context draw context
- primitive desired primitive
- points array of points in the primitive
- point_count number of points in the array

`kOfxDrawingPrimitiveLines` - like `GL_LINES`, `n` points draws `n/2` separated lines
`kOfxDrawingPrimitiveLineStrip` - like `GL_LINE_STRIP`, `n` points draws `n-1` connected lines
`kOfxDrawingPrimitiveLineLoop` - like `GL_LINE_LOOP`, `n` points draws `n` connected lines
`kOfxDrawingPrimitiveRectangle` - draws an axis-aligned filled rectangle defined by 2 opposite corner points
`kOfxDrawingPrimitivePolygon` - like `GL_POLYGON`, draws a filled `n`-sided polygon
`kOfxDrawingPrimitiveEllipse` - draws a axis-aligned elliptical line (not filled) within the rectangle defined by 2 opposite corner points

Return

- `kOfxStatOK` - the draw was completed
- `kOfxStatErrValue` - invalid primitive, or `point_count` not valid for primitive
- `kOfxStatFailed` - failure, e.g. if function is called outside `kOfxInteractActionDraw`

`OfxStatus` (***drawText**)(`OfxDrawingContextHandle` context, const char *text, const `OfxPointD` *pos, int alignment)

Draws text at the specified position.

- `context` draw context
- `text` text to draw (UTF-8 encoded)
- `pos` position at which to align the text
- `alignment` text alignment flags (see `kOfxDrawingTextAlignment*`)

The text font face and size are determined by the host.

Return

- `kOfxStatOK` - the text was drawn
- `kOfxStatErrValue` - text or pos were not defined
- `kOfxStatFailed` - failure, e.g. if function is called outside `kOfxInteractActionDraw`

struct **OfxHost**

`#include <ofxCore.h>` Generic host structure passed to `OfxPlugin::setHost` function.

This structure contains what is needed by a plug-in to bootstrap its connection to the host.

Public Members

`OfxPropertySetHandle` **host**

Global handle to the host. Extract relevant host properties from this. This pointer will be valid while the binary containing the plug-in is loaded.

const void *(***fetchSuite**)(`OfxPropertySetHandle` host, const char *suiteName, int suiteVersion)

The function which the plug-in uses to fetch suites from the host.

- `host` the host the suite is being fetched from this *must* be the `host` member of the `OfxHost` struct containing `fetchSuite`.
- `suiteName` ASCII string labelling the host supplied API

- `suiteVersion` version of that suite to fetch

Any API fetched will be valid while the binary containing the plug-in is loaded.

Repeated calls to `fetchSuite` with the same parameters will return the same pointer.

It is recommended that hosts should return the same host and suite pointers to all plugins in the same shared lib or bundle.

returns

- NULL if the API is unknown (either the api or the version requested),
- pointer to the relevant API if it was found

struct **OfxImageEffectOpenGLRenderSuiteV1**

#include <ofxGPURender.h> OFX suite that provides image to texture conversion for OpenGL processing.

Public Members

OfxStatus (***clipLoadTexture**)(*OfxImageClipHandle* clip, *OfxTime* time, const char *format, const *OfxRectD* *region, *OfxPropertySetHandle* *textureHandle)

loads an image from an OFX clip as a texture into OpenGL

- `clip` clip to load the image from
- `time` effect time to load the image from
- `format` requested texture format (As in none,byte,word,half,float, etc..) When set to NULL, the host decides the format based on the plug-in's *kOfxOpenGLPropPixelDepth* setting.
- `region` region of the image to load (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- `textureHandle` property set containing information about the texture

An image is fetched from a clip at the indicated time for the given region and loaded into an OpenGL texture. When a specific format is requested, the host ensures it gives the requested format. When the clip specified is the "Output" clip, the format is ignored and the host must bind the resulting texture as the current color buffer (render target). This may also be done prior to calling the *kOfxImageEffectActionRender* action. If the *region* parameter is set to non-NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set or is NULL, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped to the clip's Region of Definition. Information about the texture, including the texture index, is returned in the *textureHandle* argument. The properties on this handle will be...

- *kOfxImageEffectPropOpenGLTextureIndex*
- *kOfxImageEffectPropOpenGLTextureTarget*
- *kOfxImageEffectPropPixelDepth*
- *kOfxImageEffectPropComponents*
- *kOfxImageEffectPropPreMultiplication*
- *kOfxImageEffectPropRenderScale*
- *kOfxImagePropPixelAspectRatio*

- *kOfxImagePropBounds*
- *kOfxImagePropRegionOfDefinition*
- *kOfxImagePropRowBytes*
- *kOfxImagePropField*
- *kOfxImagePropUniqueIdentifier*

With the exception of the OpenGL specifics, these properties are the same as the properties in an image handle returned by `clipGetImage` in the image effect suite.

Note:

- this is the OpenGL equivalent of `clipGetImage` from *OfxImageEffectSuiteV1*
-

Pre

- clip was returned by `clipGetHandle`
- Format property in the texture handle

Post

- texture handle to be disposed of by `clipFreeTexture` before the action returns
- when the clip specified is the “Output” clip, the format is ignored and the host must bind the resulting texture as the current color buffer (render target). This may also be done prior to calling the render action.

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time and/or region, the plug-in should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - not enough OpenGL memory was available for the effect to load the texture. The plug-in should abort the GL render and return *kOfxStatErrMemory*, after which the host can decide to retry the operation with CPU based processing.

OfxStatus (***clipFreeTexture**)(*OfxPropertySetHandle* textureHandle)

Releases the texture handle previously returned by `clipLoadTexture`.

For input clips, this also deletes the texture from OpenGL. This should also be called on the output clip; for the Output clip, it just releases the handle but does not delete the texture (since the host will need to read it).

Pre

- textureHandle was returned by `clipGetImage`

Post

- all operations on textureHandle will be invalid, and the OpenGL texture it referred to has been deleted (for source clips)

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - general failure for some reason,
- *kOfxStatErrBadHandle* - the image handle was invalid,

OfxStatus (***flushResources**())

Request the host to minimize its GPU resource load.

When a plug-in fails to allocate GPU resources, it can call this function to request the host to flush its GPU resources if it holds any. After the function the plug-in can try again to allocate resources which then might succeed if the host actually has released anything.

Pre

Post

- No changes to the plug-in GL state should have been made.

Return

- *kOfxStatOK* - the host has actually released some resources,
- *kOfxStatReplyDefault* - nothing the host could do..

struct **OfxImageEffectSuiteV1**

#include <ofxImageEffect.h> The OFX suite for image effects.

This suite provides the functions needed by a plugin to defined and use an image effect plugin.

Public Members

OfxStatus (***getPropertySet**)(*OfxImageEffectHandle* imageEffect, *OfxPropertySetHandle* *propHandle)

Retrieves the property set for the given image effect.

- *imageEffect* image effect to get the property set for
- *propHandle* pointer to a the property set pointer, value is returned here

The property handle is for the duration of the image effect handle.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***getParamSet**)(*OfxImageEffectHandle* imageEffect, *OfiParamSetHandle* *paramSet)

Retrieves the parameter set for the given image effect.

- *imageEffect* image effect to get the property set for
- *paramSet* pointer to a the parameter set, value is returned here

The param set handle is valid for the lifetime of the image effect handle.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (*clipDefine)(*OfxImageEffectHandle* imageEffect, const char *name, *OfxPropertySetHandle* *propertySet)

Define a clip to the effect.

- pluginHandle handle passed into 'describeInContext' action
- name unique name of the clip to define
- propertySet property handle for the clip descriptor will be returned here

This function defines a clip to a host, the returned property set is used to describe various aspects of the clip to the host. Note that this does not create a clip instance.

Pre

- we are inside the describe in context action.

Return

OfxStatus (*clipGetHandle)(*OfxImageEffectHandle* imageEffect, const char *name, *OfxImageClipHandle* *clip, *OfxPropertySetHandle* *propertySet)

Get the property handle of the named input clip in the given instance.

- imageEffect an instance handle to the plugin
- name name of the clip, previously used in a clip define call
- clip where to return the clip
- propertySet if not NULL, the descriptor handle for a parameter's property set will be placed here.

The propertySet will have the same value as would be returned by *OfxImageEffectSuiteV1::clipGetPropertySet*

This **return** a clip handle **for** the given instance, note that this will **not** be the same **as** the clip handle returned by clipDefine **and** will be distant to clip handles **in any** other instance of the plugin.

Not a valid call **in any** of the describe actions.

Pre

- create instance action called,
- name passed to clipDefine for this context,
- not inside describe or describe in context actions.

Post

- handle will be valid for the life time of the instance.

OfxStatus (*clipGetPropertySet)(*OfxImageClipHandle* clip, *OfxPropertySetHandle* *propHandle)

Retrieves the property set for a given clip.

- clip clip effect to get the property set for
- propHandle pointer to a the property set handle, value is returned here

The property handle is valid for the lifetime of the clip, which is generally the lifetime of the instance.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (*clipGetImage)(*OfxImageClipHandle* clip, *OfxTime* time, const *OfxRectD* *region, *OfxPropertySetHandle* *imageHandle)

Get a handle for an image in a clip at the indicated time and indicated region.

- clip clip to extract the image from
- time time to fetch the image at
- region region to fetch the image from (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- imageHandle property set containing the image's data

An image is fetched from a clip at the indicated time for the given region and returned in the imageHandle.

If the *region* parameter is not set to NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped to the clip's Region of Definition.

If clipGetImage is called twice with the same parameters, then two separate image handles will be returned, each of which must be released. The underlying implementation could share image data pointers and use reference counting to maintain them.

Pre

- clip was returned by clipGetHandle

Post

- image handle is only valid for the duration of the action clipGetImage is called in
- image handle to be disposed of by clipReleaseImage before the action returns

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time and/or region, the plugin should continue operation, but assume the image was black and transparent.

- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - the host had not enough memory to complete the operation, plugin should abort whatever it was doing.

OfxStatus (*clipReleaseImage)(*OfxPropertySetHandle* imageHandle)

Releases the image handle previously returned by clipGetImage.

Pre

- imageHandle was returned by clipGetImage

Post

- all operations on imageHandle will be invalid

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatErrBadHandle* - the image handle was invalid,

OfxStatus (*clipGetRegionOfDefinition)(*OfxImageClipHandle* clip, *OfxTime* time, *OfxRectD* *bounds)

Returns the spatial region of definition of the clip at the given time.

- clipHandle clip to extract the image from
- time time to fetch the image at
- region region to fetch the image from (optional, set to NULL to get a 'default' region) this is in the CanonicalCoordinates.
- imageHandle handle where the image is returned

An image is fetched from a clip at the indicated time for the given region and returned in the imageHandle.

If the *region* parameter is not set to NULL, then it will be clipped to the clip's Region of Definition for the given time. The returned image will be *at least* as big as this region. If the region parameter is not set, then the region fetched will be at least the Region of Interest the effect has previously specified, clipped the clip's Region of Definition.

Pre

- clipHandle was returned by clipGetHandle

Post

- bounds will be filled the RoD of the clip at the indicated time

Return

- *kOfxStatOK* - the image was successfully fetched and returned in the handle,
- *kOfxStatFailed* - the image could not be fetched because it does not exist in the clip at the indicated time, the plugin should continue operation, but assume the image was black and transparent.
- *kOfxStatErrBadHandle* - the clip handle was invalid,
- *kOfxStatErrMemory* - the host had not enough memory to complete the operation, plugin should abort whatever it was doing.

int (***abort**)(*OfxImageEffectHandle* imageEffect)

Returns whether to abort processing or not.

- *imageEffect* instance of the image effect

A host may want to signal to a plugin that it should stop whatever rendering it is doing and start again. Generally this is done in interactive threads in response to users tweaking some parameter.

This function indicates whether a plugin should stop whatever processing it is doing.

Return

- 0 if the effect should continue whatever processing it is doing
- 1 if the effect should abort whatever processing it is doing

OfxStatus (***imageMemoryAlloc**)(*OfxImageEffectHandle* instanceHandle, size_t nBytes, *OfxImageMemoryHandle* *memoryHandle)

Allocate memory from the host's image memory pool.

- *instanceHandle* effect instance to associate with this memory allocation, may be NULL.
- *nBytes* number of bytes to allocate
- *memoryHandle* pointer to the memory handle where a return value is placed

Memory handles allocated by this should be freed by *OfxImageEffectSuiteV1::imageMemoryFree*. To access the memory behind the handle you need to call *OfxImageEffectSuiteV1::imageMemoryLock*.

See ImageEffectsMemoryAllocation.

Return

- *kOfxStatOK* if all went well, a valid memory handle is placed in *memoryHandle*
- *kOfxStatErrBadHandle* if *instanceHandle* is not valid, *memoryHandle* is set to NULL
- *kOfxStatErrMemory* if there was not enough memory to satisfy the call, *memoryHandle* is set to NULL

OfxStatus (***imageMemoryFree**)(*OfxImageMemoryHandle* memoryHandle)

Frees a memory handle and associated memory.

- *memoryHandle* memory handle returned by *imageMemoryAlloc*

This function frees a memory handle and associated memory that was previously allocated via *OfxImageEffectSuiteV1::imageMemoryAlloc*

If there are outstanding locks, these are ignored and the handle and memory are freed anyway.

See ImageEffectsMemoryAllocation.

Return

- *kOfxStatOK* if the memory was cleanly deleted
- *kOfxStatErrBadHandle* if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*

OfxStatus (***imageMemoryLock**)(*OfxImageMemoryHandle* memoryHandle, void **returnedPtr)

Lock the memory associated with a memory handle and make it available for use.

- *memoryHandle* memory handle returned by *imageMemoryAlloc*
- *returnedPtr* where to the pointer to the locked memory

This function locks them memory associated with a memory handle and returns a pointer to it. The memory will be 16 byte aligned, to allow use of vector operations.

Note that memory locks and unlocks nest.

After the first lock call, the contents of the memory pointer to by *returnedPtr* is undefined. All subsequent calls to lock will return memory with the same contents as the previous call.

Also, if unlocked, then relocked, the memory associated with a memory handle may be at a different address.

See also *OfxImageEffectSuiteV1::imageMemoryUnlock* and *ImageEffectsMemoryAllocation*.

Return

- *kOfxStatOK* if the memory was locked, a pointer is placed in *returnedPtr*
- *kOfxStatErrBadHandle* if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*, null is placed in **returnedPtr*
- *kOfxStatErrMemory* if there was not enough memory to satisfy the call, **returnedPtr* is set to NULL

OfxStatus (***imageMemoryUnlock**)(*OfxImageMemoryHandle* memoryHandle)

Unlock allocated image data.

- *allocatedData* pointer to memory previously returned by *OfxImageEffectSuiteV1::imageAlloc*

This function unlocks a previously locked memory handle. Once completely unlocked, memory associated with a *memoryHandle* is no longer available for use. Attempting to use it results in undefined behaviour.

Note that locks and unlocks nest, and to fully unlock memory you need to match the count of locks placed upon it.

Also note, if you unlock a completely unlocked handle, it has no effect (ie: the lock count can't be negative).

If unlocked, then relocked, the memory associated with a memory handle may be at a different address, however the contents will remain the same.

See also *OfxImageEffectSuiteV1::imageMemoryLock* and *ImageEffectsMemoryAllocation*.

Return

- *kOfxStatOK* if the memory was unlocked cleanly,
- *kOfxStatErrBadHandle* if the value of *memoryHandle* was not a valid pointer returned by *OfxImageEffectSuiteV1::imageMemoryAlloc*, null is placed in **returnedPtr*

struct **OfxInteractSuiteV1**

#include <ofxInteract.h> OFX suite that allows an effect to interact with an OpenGL window so as to provide custom interfaces.

Public Members

OfxStatus (***interactSwapBuffers**)(*OfxInteractHandle* interactInstance)

Requests an OpenGL buffer swap on the interact instance.

OfxStatus (***interactRedraw**)(*OfxInteractHandle* interactInstance)

Requests a redraw of the interact instance.

OfxStatus (***interactGetPropertySet**)(*OfxInteractHandle* interactInstance, *OfxPropertySetHandle* *property)

Gets the property set handle for this interact handle.

struct **OfxMemorySuiteV1**

#include <ofxMemory.h> The OFX suite that implements general purpose memory management.

Use this suite for ordinary memory management functions, where you would normally use malloc/free or new/delete on ordinary objects.

For images, you should use the memory allocation functions in the image effect suite, as many hosts have specific image memory pools.

Note: C++ plugin developers will need to redefine new and delete as skins ontop of this suite.

Public Members

OfxStatus (***memoryAlloc**)(void *handle, size_t nBytes, void **allocatedData)

Allocate memory.

- **handle** - effect instance to associate with this memory allocation, or NULL.
- **nBytes** number of bytes to allocate
- **allocatedData** pointer to the return value. Allocated memory will be aligned for any use.

This function has the host allocate memory using its own memory resources and returns that to the plugin.

Return

- *kOfxStatOK* the memory was successfully allocated
- *kOfxStatErrMemory* the request could not be met and no memory was allocated

OfxStatus (***memoryFree**)(void *allocatedData)

Frees memory.

- **allocatedData** pointer to memory previously returned by *OfxMemorySuiteV1::memoryAlloc*

This function frees any memory that was previously allocated via *OfxMemorySuiteV1::memoryAlloc*.

Return

- *kOfxStatOK* the memory was successfully freed
- *kOfxStatErrBadHandle* *allocatedData* was not a valid pointer returned by *OfxMemorySuiteV1::memoryAlloc*

struct **OfxMessageSuiteV1**

#include <ofxMessage.h> The OFX suite that allows a plug-in to pass messages back to a user. The V2 suite extends on this in a backwards compatible manner.

Public Members

OfxStatus (***message**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a message on the host, using printf style varargs.

- *handle* effect handle (descriptor or instance) the message should be associated with, may be NULL
- *messageType* string describing the kind of message to post, one of the *kOfxMessageType** constants
- *messageId* plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- *format* printf style format string
- ... printf style varargs list to print

Return

- *kOfxStatOK* - if the message was successfully posted
- *kOfxStatReplyYes* - if the message was of type *kOfxMessageQuestion* and the user reply yes
- *kOfxStatReplyNo* - if the message was of type *kOfxMessageQuestion* and the user reply no
- *kOfxStatFailed* - if the message could not be posted for some reason

struct **OfxMessageSuiteV2**

#include <ofxMessage.h> The OFX suite that allows a plug-in to pass messages back to a user.

This extends *OfxMessageSuiteV1*, and should be considered a replacement to version 1.

Note that this suite has been extended in backwards compatible manner, so that a host can return this struct for both V1 and V2.

Public Members

OfxStatus (***message**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a transient message on the host, using printf style varargs. Same as the V1 message suite call.

- **handle** effect handle (descriptor or instance) the message should be associated with, may be null
- **messageType** string describing the kind of message to post, one of the `kOfxMessageType*` constants
- **messageId** plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- **format** printf style format string
- ... printf style varargs list to print

Return

- *kOfxStatOK* - if the message was successfully posted
- *kOfxStatReplyYes* - if the message was of type `kOfxMessageQuestion` and the user reply yes
- *kOfxStatReplyNo* - if the message was of type `kOfxMessageQuestion` and the user reply no
- *kOfxStatFailed* - if the message could not be posted for some reason

OfxStatus (***setPersistentMessage**)(void *handle, const char *messageType, const char *messageId, const char *format, ...)

Post a persistent message on an effect, using printf style varargs, and set error states. New for V2 message suite.

- **handle** effect instance handle the message should be associated with, may NOT be null,
- **messageType** string describing the kind of message to post, should be one of ...
 - `kOfxMessageError`
 - `kOfxMessageWarning`
 - `kOfxMessageMessage`
- **messageId** plugin specified id to associate with this message. If overriding the message in XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur,
- **format** printf style format string
- ... printf style varargs list to print

Persistent messages are associated with an effect handle until explicitly cleared by an effect. So if an error message is posted the error state, and associated message will persist and be displayed on the effect appropriately. (eg: draw a node in red on a node based compositor and display the message when clicked on).

If *messageType* is error or warning, associated error states should be flagged on host applications. Posting an error message implies that the host cannot proceed, a warning allows the host to proceed, whilst a simple message should have no stop anything.

Return

- *kOfxStatOK* - if the message was successfully posted
- *kOfxStatErrBadHandle* - the handle was rubbish
- *kOfxStatFailed* - if the message could not be posted for some reason

OfxStatus (*clearPersistentMessage)(void *handle)

Clears any persistent message on an effect handle that was set by *OfxMessageSuiteV2::setPersistentMessage*. New for V2 message suite.

- handle effect instance handle messages should be cleared from.
- handle effect handle (descriptor or instance)

Clearing a message will clear any associated error state.

Return

- *kOfxStatOK* - if the message was successfully cleared
- *kOfxStatErrBadHandle* - the handle was rubbish
- *kOfxStatFailed* - if the message could not be cleared for some reason

struct **OfxMultiThreadSuiteV1**

#include <ofxMultiThread.h> OFX suite that provides simple SMP style multi-processing.

Public Members

OfxStatus (*multiThread)(OfxThreadFunctionV1 func, unsigned int nThreads, void *customArg)

Function to spawn SMP threads.

- func function to call in each thread.
- nThreads number of threads to launch
- customArg paramter to pass to customArg of func in each thread.

This function will spawn nThreads separate threads of computation (typically one per CPU) to allow something to perform symmetric multi processing. Each thread will call 'func' passing in the index of the thread and the number of threads actually launched.

multiThread will not return until all the spawned threads have returned. It is up to the host how it waits for all the threads to return (busy wait, blocking, whatever).

nThreads can be more than the value returned by multiThreadNumCPUs, however the threads will be limited to the number of CPUs returned by multiThreadNumCPUs.

This function cannot be called recursively.

Return

- *kOfxStatOK*, the function func has executed and returned successfully
- *kOfxStatFailed*, the threading function failed to launch
- *kOfxStatErrExists*, failed in an attempt to call multiThread recursively,

OfxStatus (***multiThreadNumCPUs**)(unsigned int *nCPUs)

Function which indicates the number of CPUs available for SMP processing.

- nCPUs pointer to an integer where the result is returned

This value may be less than the actual number of CPUs on a machine, as the host may reserve other CPUs for itself.

Return

- *kOfxStatOK*, all was OK and the maximum number of threads is in nThreads.
- *kOfxStatFailed*, the function failed to get the number of CPUs

OfxStatus (***multiThreadIndex**)(unsigned int *threadIndex)

Function which indicates the index of the current thread.

- threadIndex pointer to an integer where the result is returned

This function returns the thread index, which is the same as the *threadIndex* argument passed to the *Ofx-ThreadFunctionV1*.

If there are no threads currently spawned, then this function will set threadIndex to 0

Return

- *kOfxStatOK*, all was OK and the maximum number of threads is in nThreads.
- *kOfxStatFailed*, the function failed to return an index

int (***multiThreadIsSpawnedThread**)(void)

Function to enquire if the calling thread was spawned by multiThread.

Return

- 0 if the thread is not one spawned by multiThread
- 1 if the thread was spawned by multiThread

OfxStatus (***mutexCreate**)(*OfxMutexHandle* *mutex, int lockCount)

Create a mutex.

- mutex where the new handle is returned
- count initial lock count on the mutex. This can be negative.

Creates a new mutex with lockCount locks on the mutex intially set.

Return

- *kOfxStatOK* - mutex is now valid and ready to go

OfxStatus (***mutexDestroy**)(const *OfxMutexHandle* mutex)

Destroy a mutex.

Destroys a mutex intially created by mutexCreate.

Return

- kOfxStatOK - if it destroyed the mutex
- kOfxStatErrBadHandle - if the handle was bad

OfxStatus (***mutexLock**)(const *OfxMutexHandle* mutex)

Blocking lock on the mutex.

This tries to lock a mutex and blocks the thread it is in until the lock succeeds.

A successful lock causes the mutex's lock count to be increased by one and to block any other calls to lock the mutex until it is unlocked.

Return

- kOfxStatOK - if it got the lock
- kOfxStatErrBadHandle - if the handle was bad

OfxStatus (***mutexUnLock**)(const *OfxMutexHandle* mutex)

Unlock the mutex.

This unlocks a mutex. Unlocking a mutex decreases its lock count by one.

Return

- kOfxStatOK if it released the lock
- kOfxStatErrBadHandle if the handle was bad

OfxStatus (***mutexTryLock**)(const *OfxMutexHandle* mutex)

Non blocking attempt to lock the mutex.

This attempts to lock a mutex, if it cannot, it returns and says so, rather than blocking.

A successful lock causes the mutex's lock count to be increased by one, if the lock did not succeed, the call returns immediately and the lock count remains unchanged.

Return

- kOfxStatOK - if it got the lock
- kOfxStatFailed - if it did not get the lock
- kOfxStatErrBadHandle - if the handle was bad

struct **OfxOpenCLProgramSuiteV1**

#include <ofxGPURender.h> OFX suite that allows a plug-in to get OpenCL programs compiled.

This is an optional suite the host can provide for building OpenCL programs for the plug-in, as an alternative to calling `clCreateProgramWithSource` / `clBuildProgram`. There are two advantages to doing this: The host can add flags (such as `-cl-denorms-are-zero`) to the build call, and may also cache program binaries for performance (however, if the source of the program or the OpenCL environment changes, the host must recompile so some mechanism such as hashing must be used).

Public Members

OfxStatus (***compileProgram**)(const char *pszProgramSource, int fOptional, void *pResult)

Compiles the OpenCL program.

struct **OfxParameterSuiteV1**

#include <ofxParam.h> The OFX suite used to define and manipulate user visible parameters.

Keyframe Handling

These functions allow the plug-in to delete and get information about keyframes.

To set keyframes, use *paramSetValueAtTime()*.

paramGetKeyTime and *paramGetKeyIndex* use indices to refer to keyframes. Keyframes are stored by the host in increasing time order, so $\text{time}(\text{kf}[i]) < \text{time}(\text{kf}[i+1])$. Keyframe indices will change whenever keyframes are added, deleted, or moved in time, whether by the host or by the plug-in. They may vary between actions if the user changes a keyframe. The keyframe indices will not change within a single action.

OfxStatus (***paramGetNumKeys**)(*OfxParamHandle* paramHandle, unsigned int *numberOfKeys)

Returns the number of keyframes in the parameter.

- *paramHandle* parameter handle to interrogate
- *numberOfKeys* pointer to integer where the return value is placed

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Returns the number of keyframes in the parameter.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetKeyTime**)(*OfxParamHandle* paramHandle, unsigned int nthKey, *OfxTime* *time)

Returns the time of the nth key.

- *paramHandle* parameter handle to interrogate
- *nthKey* which key to ask about (0 to *paramGetNumKeys* - 1), ordered by time
- *time* pointer to *OfxTime* where the return value is placed

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrBadIndex* - the nthKey does not exist

OfxStatus (***paramGetKeyIndex**)(*OfxParamHandle* paramHandle, *OfxTime* time, int direction, int *index)

Finds the index of a keyframe at/before/after a specified time.

- **paramHandle** parameter handle to search
- **time** what time to search from
- **direction**
 - == 0 indicates search for a key at the indicated time (some small delta)
 - > 0 indicates search for the next key after the indicated time
 - < 0 indicates search for the previous key before the indicated time
- **index** pointer to an integer which in which the index is returned set to -1 if no key was found

Return

- *kOfxStatOK* - all was OK
- *kOfxStatFailed* - if the search failed to find a key
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramDeleteKey**)(*OfxParamHandle* paramHandle, *OfxTime* time)

Deletes a keyframe if one exists at the given time.

- **paramHandle** parameter handle to delete the key from
- **time** time at which a keyframe is

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrBadIndex* - no key at the given time

OfxStatus (***paramDeleteAllKeys**)(*OfxParamHandle* paramHandle)

Deletes all keyframes from a parameter.

- **paramHandle** parameter handle to delete the keys from
- **name** parameter to delete the keyframes from is

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

Public Members

OfxStatus (***paramDefine**)(*OfxParamSetHandle* paramSet, const char *paramType, const char *name, *OfxPropertySetHandle* *propertySet)

Defines a new parameter of the given type in a describe action.

- paramSet handle to the parameter set descriptor that will hold this parameter
- paramType type of the parameter to create, one of the *kOfxParamType** #defines
- name unique name of the parameter
- propertySet if not null, a pointer to the parameter descriptor's property set will be placed here.

This function defines a parameter in a parameter set and returns a property set which is used to describe that parameter.

This function does not actually create a parameter, it only says that one should exist in any subsequent instances. To fetch an parameter instance *paramGetHandle* must be called on an instance.

This function can always be called in one of a plug-in's "describe" functions which defines the parameter sets common to all instances of a plugin.

Return

- *kOfxStatOK* - the parameter was created correctly
- *kOfxStatErrBadHandle* - if the plugin handle was invalid
- *kOfxStatErrExists* - if a parameter of that name exists already in this plugin
- *kOfxStatErrUnknown* - if the type is unknown
- *kOfxStatErrUnsupported* - if the type is known but unsupported

OfxStatus (***paramGetHandle**)(*OfxParamSetHandle* paramSet, const char *name, *OfxParamHandle* *param, *OfxPropertySetHandle* *propertySet)

Retrieves the handle for a parameter in a given parameter set.

- paramSet instance of the plug-in to fetch the property handle from
- name parameter to ask about
- param pointer to a param handle, the value is returned here
- propertySet if not null, a pointer to the parameter's property set will be placed here.

Parameter handles retrieved from an instance are always distinct in each instance. The parameter handle is valid for the life-time of the instance. Parameter handles in instances are distinct from parameter handles in plugins. You cannot call this in a plugin's describe function, as it needs an instance to work on.

Return

- *kOfxStatOK* - the parameter was found and returned
- *kOfxStatErrBadHandle* - if the plugin handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramSetGetPropertySet**)(*OfiParamSetHandle* paramSet, *OfxPropertySetHandle* *propHandle)

Retrieves the property set handle for the given parameter set.

- paramSet parameter set to get the property set for
- propHandle pointer to a the property set handle, value is returned here

Note: The property handle belonging to a parameter set is the same as the property handle belonging to the plugin instance.

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramGetPropertySet**)(*OfiParamHandle* param, *OfxPropertySetHandle* *propHandle)

Retrieves the property set handle for the given parameter.

- param parameter to get the property set for
- propHandle pointer to a the property set handle, value is returned here

The property handle is valid for the lifetime of the parameter, which is the lifetime of the instance that owns the parameter

Return

- *kOfxStatOK* - the property set was found and returned
- *kOfxStatErrBadHandle* - if the parameter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***paramGetValue**)(*OfiParamHandle* paramHandle, ...)

Gets the current value of a parameter,.

- paramHandle parameter handle to fetch value from
- ... one or more pointers to variables of the relevant type to hold the parameter's value

This gets the current value of a parameter. The varargs ... argument needs to be pointer to C variables of the relevant type for this parameter. Note that params with multiple values (eg Colour) take multiple args here. For example...

```
OfiParamHandle myDoubleParam, *myColourParam;
ofxHost->paramGetHandle(instance, "myDoubleParam", &myDoubleParam);
double myDoubleValue;
ofxHost->paramGetValue(myDoubleParam, &myDoubleValue);
```

(continues on next page)

(continued from previous page)

```

ofxHost->paramGetHandle(instance, "myColourParam", &myColourParam);
double myR, myG, myB;
ofxHost->paramGetValue(myColourParam, &myR, &myG, &myB);

```

Note: `paramGetValue` should only be called from within a *kOfxActionInstanceChanged* or interact action and never from the render actions (which should always use `paramGetValueAtTime`).

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetValueAtTime**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Gets the value of a parameter at a specific time.

- `paramHandle` parameter handle to fetch value from
- `time` at what point in time to look up the parameter
- ... one or more pointers to variables of the relevant type to hold the parameter's value

This gets the current value of a parameter. The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteVI::paramGetValue* for notes on the varags list

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetDerivative**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Gets the derivative of a parameter at a specific time.

- `paramHandle` parameter handle to fetch value from
- `time` at what point in time to look up the parameter
- ... one or more pointers to variables of the relevant type to hold the parameter's derivative

This gets the derivative of the parameter at the indicated time.

The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteVI::paramGetValue* for notes on the varags list.

Only double and colour params can have their derivatives found.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramGetIntegral**)(*OfxParamHandle* paramHandle, *OfxTime* time1, *OfxTime* time2, ...)

Gets the integral of a parameter over a specific time range,.

- **paramHandle** parameter handle to fetch integral from
- **time1** where to start evaluating the integral
- **time2** where to stop evaluating the integral
- ... one or more pointers to variables of the relevant type to hold the parameter's integral

This gets the integral of the parameter over the specified time range.

The varargs needs to be pointer to C variables of the relevant type for this parameter. See *OfxParameterSuiteV1::paramGetValue* for notes on the varargs list.

Only double and colour params can be integrated.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramSetValue**)(*OfxParamHandle* paramHandle, ...)

Sets the current value of a parameter.

- **paramHandle** parameter handle to set value in
- ... one or more variables of the relevant type to hold the parameter's value

This sets the current value of a parameter. The varargs ... argument needs to be values of the relevant type for this parameter. Note that params with multiple values (eg Colour) take multiple args here. For example...

```
ofxHost->paramSetValue(instance, "myDoubleParam", double(10));
ofxHost->paramSetValue(instance, "myColourParam", double(pix.r), double(pix.
↪g), double(pix.b));
```

Note: *paramSetValue* should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramSetValueAtTime**)(*OfxParamHandle* paramHandle, *OfxTime* time, ...)

Keyframes the value of a parameter at a specific time.

- **paramHandle** parameter handle to set value in
- **time** at what point in time to set the keyframe

- ... one or more variables of the relevant type to hold the parameter's value

This sets a keyframe in the parameter at the indicated time to have the indicated value. The varargs ... argument needs to be values of the relevant type for this parameter. See the note on *OfxParameterSuiteV1::paramSetValue* for more detail

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Note: *paramSetValueAtTime* should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramCopy**)(*OfxParamHandle* paramTo, *OfxParamHandle* paramFrom, *OfxTime* dstOffset, const *OfxRangeD* *frameRange)

Copies one parameter to another, including any animation etc...

- *paramTo* parameter to set
- *paramFrom* parameter to copy from
- *dstOffset* temporal offset to apply to keys when writing to the *paramTo*
- *frameRange* if *paramFrom* has animation, and *frameRange* is not null, only this range of keys will be copied

This copies the value of *paramFrom* to *paramTo*, including any animation it may have. All the previous values in *paramTo* will be lost.

To choose all animation in *paramFrom* set *frameRange* to [0, 0]

V1.3: This function can be called the *kOfxActionInstanceChanged* action and during image effect analysis render passes. V1.4: This function can be called the *kOfxActionInstanceChanged* action

Pre

- Both parameters must be of the same type.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the parameter handle was invalid

OfxStatus (***paramEditBegin**)(*OfxParamSetHandle* paramSet, const char *name)

Used to group any parameter changes for undo/redo purposes.

- *paramSet* the parameter set in which this is happening
- *name* label to attach to any undo/redo string UTF8

If a plugin calls `paramSetValue/paramSetValueAtTime` on one or more parameters, either from custom GUI interaction or some analysis of imagery etc.. this is used to indicate the start of a set of a parameter changes that should be considered part of a single undo/redo block.

See also *OfxParameterSuiteV1::paramEditEnd*

Note: `paramEditBegin` should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the instance handle was invalid

OfxStatus (***paramEditEnd**)(*OfxParamSetHandle* paramSet)

Used to group any parameter changes for undo/redo purposes.

- `paramSet` parameter set in which this is happening

If a plugin calls `paramSetValue/paramSetValueAtTime` on one or more parameters, either from custom GUI interaction or some analysis of imagery etc.. this is used to indicate the end of a set of parameter changes that should be considered part of a single undo/redo block

See also *OfxParameterSuiteV1::paramEditBegin*

Note: `paramEditEnd` should only be called from within a *kOfxActionInstanceChanged* or interact action.

Return

- *kOfxStatOK* - all was OK
- *kOfxStatErrBadHandle* - if the instance handle was invalid

struct **OfxParametricParameterSuiteV1**

#include <ofxParametricParam.h> The OFX suite used to define and manipulate ‘parametric’ parameters.

This is an optional suite.

Parametric parameters are in effect ‘functions’ a plug-in can ask a host to arbitrarily evaluate for some value ‘x’. A classic use case would be for constructing look-up tables, a plug-in would ask the host to evaluate one at multiple values from 0 to 1 and use that to fill an array.

A host would probably represent this to a user as a cubic curve in a standard curve editor interface, or possibly through scripting. The user would then use this to define the ‘shape’ of the parameter.

The evaluation of such params is not the same as animation, they are returning values based on some arbitrary argument orthogonal to time, so to evaluate such a param, you need to pass a parametric position and time.

Often, you would want such a parametric parameter to be multi-dimensional, for example, a colour look-up table might want three values, one for red, green and blue. Rather than declare three separate parametric parameters, it would be better to have one such parameter with multiple values in it.

The major complication with these parameters is how to allow a plug-in to set values, and defaults. The default default value of a parametric curve is to be an identity lookup. If a plugin wishes to set a different default value for a curve, it can use the suite to set key/value pairs on the *descriptor* of the param. When a new instance is made, it will have these curve values as a default.

Public Members

OfxStatus (***parametricParamGetValue**)(*OfxParamHandle* param, int curveIndex, *OfxTime* time, double parametricPosition, double *returnValue)

Evaluates a parametric parameter.

- param handle to the parametric parameter
- curveIndex which dimension to evaluate
- time the time to evaluate to the parametric param at
- parametricPosition the position to evaluate the parametric param at
- returnValue pointer to a double where a value is returned

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrBadIndex* - the curve index was invalid

OfxStatus (***parametricParamGetNControlPoints**)(*OfxParamHandle* param, int curveIndex, double time, int *returnValue)

Returns the number of control points in the parametric param.

- param handle to the parametric parameter
- curveIndex which dimension to check
- time the time to check
- returnValue pointer to an integer where the value is returned.

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrBadIndex* - the curve index was invalid

OfxStatus (***parametricParamGetNthControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, int nthCtl, double *key, double *value)

Returns the key/value pair of the nth control point.

- param handle to the parametric parameter
- curveIndex which dimension to check
- time the time to check
- nthCtl the nth control point to get the value of
- key pointer to a double where the key will be returned
- value pointer to a double where the value will be returned

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***parametricParamSetNthControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, int nthCtl, double key, double value, bool addAnimationKey)

Modifies an existing control point on a curve.

- param handle to the parametric parameter
- curveIndex which dimension to set
- time the time to set the value at
- nthCtl the control point to modify
- key key of the control point
- value value of the control point
- addAnimationKey if the param is an animatable, setting this to true will force an animation keyframe to be set as well as a curve key, otherwise if false, a key will only be added if the curve is already animating.

This modifies an existing control point. Note that by changing key, the order of the control point may be modified (as you may move it before or after another point). So be careful when iterating over a curves control points and you change a key.

Return

- *kOfxStatOK* - all was fine
- *kOfxStatErrBadHandle* - if the paramter handle was invalid
- *kOfxStatErrUnknown* - if the type is unknown

OfxStatus (***parametricParamAddControlPoint**)(*OfxParamHandle* param, int curveIndex, double time, double key, double value, bool addAnimationKey)

Adds a control point to the curve.

- `param` handle to the parametric parameter
- `curveIndex` which dimension to set
- `time` the time to set the value at
- `key` key of the control point
- `value` value of the control point
- `addAnimationKey` if the `param` is an animatable, setting this to true will force an animation keyframe to be set as well as a curve key, otherwise if false, a key will only be added if the curve is already animating.

This will add a new control point to the given dimension of a parametric parameter. If a key exists sufficiently close to 'key', then it will be set to the indicated control point.

Return

- `kOfxStatOK` - all was fine
- `kOfxStatErrBadHandle` - if the parameter handle was invalid
- `kOfxStatErrUnknown` - if the type is unknown

OfxStatus (***parametricParamDeleteControlPoint**)(*OfxParamHandle* param, int curveIndex, int nthCtl)

Deletes the nth control point from a parametric param.

- `param` handle to the parametric parameter
- `curveIndex` which dimension to delete
- `nthCtl` the control point to delete

OfxStatus (***parametricParamDeleteAllControlPoints**)(*OfxParamHandle* param, int curveIndex)

Delete all curve control points on the given param.

- `param` handle to the parametric parameter
- `curveIndex` which dimension to clear

struct **OfxPlugin**

#include <ofxCore.h> The structure that defines a plug-in to a host.

This structure is the first element in any plug-in structure using the OFX plug-in architecture. By examining its members a host can determine the API that the plug-in implements, the version of that API, its name and version.

For details see Architecture.

Public Members

const char ***pluginApi**

Defines the type of the plug-in, this will tell the host what the plug-in does. e.g.: an image effects plug-in would be a “OfxImageEffectPlugin”

int **apiVersion**

Defines the version of the pluginApi that this plug-in implements

const char ***pluginIdentifier**

String that uniquely labels the plug-in among all plug-ins that implement an API. It need not necessarily be human sensible, however the preference is to use reverse internet domain name of the developer, followed by a ‘.’ then by a name that represents the plug-in.. It must be a legal ASCII string and have no whitespace in the name and no non printing chars. For example “uk.co.somesoftwarehouse.myPlugin”

unsigned int **pluginVersionMajor**

Major version of this plug-in, this gets incremented when backwards compatibility is broken.

unsigned int **pluginVersionMinor**

Major version of this plug-in, this gets incremented when software is changed, but does not break backwards compatibility.

void (***setHost**)(OfxHost *host)

Function the host uses to connect the plug-in to the host’s api fetcher.

- **fetchApi** pointer to host’s API fetcher

Mandatory function.

The very first function called in a plug-in. The plug-in *must not* call any OFX functions within this, it must only set its local copy of the host pointer.

It is recommended that hosts should return the same host and suite pointers to all plugins in the same shared lib or bundle.

Pre

- nothing else has been called

Post

- the pointer suite is valid until the plug-in is unloaded

OfxPluginEntryPoint ***mainEntry**

Main entry point for plug-ins.

Mandatory function.

The exact set of actions is determined by the plug-in API that is being implemented, however all plug-ins can perform several actions. For the list of actions consult OFX Actions.

Preconditions

- setHost has been called

struct **OfxPointD**

#include <ofxCore.h> Defines two dimensional double point.

Public Members

double **x**

double **y**

struct **OfxPointI**

#include <ofxCore.h> Defines two dimensional integer point.

Public Members

int **x**

int **y**

struct **OfxProgressSuiteV1**

#include <ofxProgress.h> A suite that provides progress feedback from a plugin to an application.

A plugin instance can initiate, update and close a progress indicator with this suite.

This is an optional suite in the Image Effect API.

API V1.4: Amends the documentation of progress suite V1 so that it is expected that it can be raised in a modal manner and have a “cancel” button when invoked in instanceChanged. Plugins that perform analysis post an appropriate message, raise the progress monitor in a modal manner and should poll to see if processing has been aborted. Any cancellation should be handled gracefully by the plugin (eg: reset analysis parameters to default values), clear allocated memory...

Many hosts already operate as described above. kOfxStatReplyNo should be returned to the plugin during progressUpdate when the user presses cancel.

Suite V2: Adds an ID that can be looked up for internationalisation and so on. When a new version is introduced, because plug-ins need to support old versions, and plug-in’s new releases are not necessary in synch with hosts (or users don’t immediately update), best practice is to support the 2 suite versions. That is, the plugin should check if V2 exists; if not then check if V1 exists. This way a graceful transition is guaranteed. So plugin should fetchSuite passing 2, (OfxProgressSuiteV2*) fetchSuite(mHost->mHost->host, kOfxProgressSuite,2); and if no success pass (OfxProgressSuiteV1*) fetchSuite(mHost->mHost->host, kOfxProgressSuite,1);

Public Members

OfxStatus (***progressStart**)(void *effectInstance, const char *label)

Initiate a progress bar display.

Call this to initiate the display of a progress bar.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.
- *label* a text label to display in any message portion of the progress object's user interface. A UTF8 string.

Pre

- There is no currently ongoing progress display for this instance.

Return

- *kOfxStatOK* - the handle is now valid for use
- *kOfxStatFailed* - the progress object failed for some reason
- *kOfxStatErrBadHandle* - *effectInstance* was invalid

OfxStatus (***progressUpdate**)(void *effectInstance, double progress)

Indicate how much of the processing task has been completed and reports on any abort status.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.
- *progress* a number between 0.0 and 1.0 indicating what proportion of the current task has been processed.

Return

- *kOfxStatOK* - the progress object was successfully updated and the task should continue
- *kOfxStatReplyNo* - the progress object was successfully updated and the task should abort
- *kOfxStatErrBadHandle* - the progress handle was invalid,

OfxStatus (***progressEnd**)(void *effectInstance)

Signal that we are finished with the progress meter.

Call this when you are done with the progress meter and no longer need it displayed.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.

Post

- you can no longer call *progressUpdate* on the instance

Return

- *kOfxStatOK* - the progress object was successfully closed
- *kOfxStatErrBadHandle* - the progress handle was invalid,

```
struct OfxProgressSuiteV2
```

```
    #include <ofxProgress.h>
```

Public Members

```
OfxStatus (*progressStart)(void *effectInstance, const char *message, const char *messageid)
```

Initiate a progress bar display.

Call this to initiate the display of a progress bar.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.
- *message* a text label to display in any message portion of the progress object's user interface. A UTF8 string.
- *messageId* plugin-specified id to associate with this message. If overriding the message in an XML resource, the message is identified with this, this may be NULL, or "", in which case no override will occur. New in V2 of this suite.

Pre

- There is no currently ongoing progress display for this instance.

Return

- *kOfxStatOK* - the handle is now valid for use
- *kOfxStatFailed* - the progress object failed for some reason
- *kOfxStatErrBadHandle* - *effectInstance* was invalid

```
OfxStatus (*progressUpdate)(void *effectInstance, double progress)
```

Indicate how much of the processing task has been completed and reports on any abort status.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.
- *progress* a number between 0.0 and 1.0 indicating what proportion of the current task has been processed.

Return

- *kOfxStatOK* - the progress object was successfully updated and the task should continue
- *kOfxStatReplyNo* - the progress object was successfully updated and the task should abort
- *kOfxStatErrBadHandle* - the progress handle was invalid,

```
OfxStatus (*progressEnd)(void *effectInstance)
```

Signal that we are finished with the progress meter.

Call this when you are done with the progress meter and no longer need it displayed.

- *effectInstance* the instance of the plugin this progress bar is associated with. It cannot be NULL.

Post

- you can no longer call `progressUpdate` on the instance

Return

- `kOfxStatOK` - the progress object was successfully closed
- `kOfxStatErrBadHandle` - the progress handle was invalid,

struct **OfxPropertySuiteV1**

#include <ofxProperty.h> The OFX suite used to access properties on OFX objects.

Public Members

OfxStatus (***propSetPointer**)(*OfxPropertySetHandle* properties, const char *property, int index, void *value)

Set a single value in a pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` for multidimensional properties and is dimension of the one we are setting
- `value` value of the property we are setting

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`
- `kOfxStatErrUnknown`
- `kOfxStatErrBadIndex`
- `kOfxStatErrValue`

OfxStatus (***propSetString**)(*OfxPropertySetHandle* properties, const char *property, int index, const char *value)

Set a single value in a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` for multidimensional properties and is dimension of the one we are setting
- `value` value of the property we are setting

Return

- `kOfxStatOK`
- `kOfxStatErrBadHandle`
- `kOfxStatErrUnknown`

- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetDouble**)(*OfxPropertySetHandle* properties, const char *property, int index, double value)

Set a single value in a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` for multidimensional properties and is dimension of the one we are setting
- `value` value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetInt**)(*OfxPropertySetHandle* properties, const char *property, int index, int value)

Set a single value in an int property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` for multidimensional properties and is dimension of the one we are setting
- `value` value of the property we are setting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetPointerN**)(*OfxPropertySetHandle* properties, const char *property, int count, void *const *value)

Set multiple values of the pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property

- count number of values we are setting in that property (ie: indices 0..count-1)
- value pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetStringN**)(*OfxPropertySetHandle* properties, const char *property, int count, const char *const *value)

Set multiple values of a string property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- count number of values we are setting in that property (ie: indices 0..count-1)
- value pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propSetDoubleN**)(*OfxPropertySetHandle* properties, const char *property, int count, const double *value)

Set multiple values of a double property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- count number of values we are setting in that property (ie: indices 0..count-1)
- value pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

- *kOfxStatErrValue*

OfxStatus (***propSetIntN**)(*OfxPropertySetHandle* properties, const char *property, int count, const int *value)

Set multiple values of an int property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are setting in that property (ie: indices 0..count-1)
- `value` pointer to an array of property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*
- *kOfxStatErrValue*

OfxStatus (***propGetPointer**)(*OfxPropertySetHandle* properties, const char *property, int index, void **value)

Get a single value from a pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetString**)(*OfxPropertySetHandle* properties, const char *property, int index, char **value)

Get a single value of a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetDouble**)(*OfxPropertySetHandle* properties, const char *property, int index, double *value)

Get a single value of a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

See the note `ArchitectureStrings` for how to deal with strings.

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetInt**)(*OfxPropertySetHandle* properties, const char *property, int index, int *value)

Get a single value of an int property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `index` refers to the index of a multi-dimensional property
- `value` pointer the return location

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propGetPointerN**)(*OfxPropertySetHandle* properties, const char *property, int count, void **value)

Get multiple values of a pointer property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indices 0..count-1)
- `value` pointer to an array of where we will return the property values

Return

- *`kOfxStatOK`*
- *`kOfxStatErrBadHandle`*
- *`kOfxStatErrUnknown`*
- *`kOfxStatErrBadIndex`*

`OfxStatus` (***propGetStringN**)(*`OfxPropertySetHandle`* `properties`, `const char *property`, `int count`, `char **value`)

Get multiple values of a string property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indices 0..count-1)
- `value` pointer to an array of where we will return the property values

See the note `ArchitectureStrings` for how to deal with strings.

Return

- *`kOfxStatOK`*
- *`kOfxStatErrBadHandle`*
- *`kOfxStatErrUnknown`*
- *`kOfxStatErrBadIndex`*

`OfxStatus` (***propGetDoubleN**)(*`OfxPropertySetHandle`* `properties`, `const char *property`, `int count`, `double *value`)

Get multiple values of a double property.

- `properties` handle of the thing holding the property
- `property` string labelling the property
- `count` number of values we are getting of that property (ie: indices 0..count-1)
- `value` pointer to an array of where we will return the property values

Return

- *`kOfxStatOK`*
- *`kOfxStatErrBadHandle`*
- *`kOfxStatErrUnknown`*

- *kOfxStatErrBadIndex*

OfxStatus (***propGetIntN**)(*OfxPropertySetHandle* properties, const char *property, int count, int *value)

Get multiple values of an int property.

- *properties* handle of the thing holding the property
- *property* string labelling the property
- *count* number of values we are getting of that property (ie: indicies 0..count-1)
- *value* pointer to an array of where we will return the property values

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*
- *kOfxStatErrBadIndex*

OfxStatus (***propReset**)(*OfxPropertySetHandle* properties, const char *property)

Resets all dimensions of a property to its default value.

- *properties* handle of the thing holding the property
- *property* string labelling the property we are resetting

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*

OfxStatus (***propGetDimension**)(*OfxPropertySetHandle* properties, const char *property, int *count)

Gets the dimension of the property.

- *properties* handle of the thing holding the property
- *property* string labelling the property we are resetting
- *count* pointer to an integer where the value is returned

Return

- *kOfxStatOK*
- *kOfxStatErrBadHandle*
- *kOfxStatErrUnknown*

struct **OfxRangeD**

#include <ofxCORE.h> Defines one dimensional double bounds.

Public Members

double **min**

double **max**

struct **OfxRangeI**

#include <ofxCORE.h> Defines one dimensional integer bounds.

Public Members

int **min**

int **max**

struct **OfxRectD**

#include <ofxCORE.h> Defines two dimensional double region.

Regions are $x_1 \leq x < x_2$

Infinite regions are flagged by setting

- $x_1 = kOfxFlagInfiniteMin$
- $y_1 = kOfxFlagInfiniteMin$
- $x_2 = kOfxFlagInfiniteMax$
- $y_2 = kOfxFlagInfiniteMax$

Public Members

double **x1**

double **y1**

double **x2**

double **y2**

struct **OfxRectI**

#include <ofxCore.h> Defines two dimensional integer region.

Regions are $x1 \leq x < x2$

Infinite regions are flagged by setting

- $x1 = kOfxFlagInfiniteMin$
- $y1 = kOfxFlagInfiniteMin$
- $x2 = kOfxFlagInfiniteMax$
- $y2 = kOfxFlagInfiniteMax$

Public Members

int **x1**

int **y1**

int **x2**

int **y2**

struct **OfxRGBAColourB**

#include <ofxPixels.h> Defines an 8 bit per component RGBA pixel.

Public Members

unsigned char **r**

unsigned char **g**

unsigned char **b**

unsigned char **a**

struct **OfxRGBAColourD**

#include <ofxPixels.h> Defines a double precision floating point component RGBA pixel.

Public Membersdouble **r**double **g**double **b**double **a**struct **OfxRGBAColourF***#include* <ofxPixels.h> Defines a floating point component RGBA pixel.**Public Members**float **r**float **g**float **b**float **a**struct **OfxRGBAColourS***#include* <ofxPixels.h> Defines a 16 bit per component RGBA pixel.**Public Members**unsigned short **r**unsigned short **g**unsigned short **b**unsigned short **a**struct **OfxRGBColourB***#include* <ofxPixels.h> Defines an 8 bit per component RGB pixel.

Public Members

unsigned char **r**

unsigned char **g**

unsigned char **b**

struct **OfxRGBColourD**

#include <ofxPixels.h> Defines a double precision floating point component RGB pixel.

Public Members

double **r**

double **g**

double **b**

struct **OfxRGBColourF**

#include <ofxPixels.h> Defines a floating point component RGB pixel.

Public Members

float **r**

float **g**

float **b**

struct **OfxRGBColourS**

#include <ofxPixels.h> Defines a 16 bit per component RGB pixel.

Public Members

unsigned short **r**

unsigned short **g**

unsigned short **b**

struct **OfxTimeLineSuiteV1**

#include <ofxTimeLine.h> Suite to control timelines.

This suite is used to enquire and control a timeline associated with a plug-in instance.

This is an optional suite in the Image Effect API.

Public Members

OfxStatus (***getTime**)(void *instance, double *time)

Get the time value of the timeline that is controlling to the indicated effect.

- *instance* is the instance of the effect changing the timeline, cast to a void *
- *time* pointer through which the timeline value should be returned

This function returns the current time value of the timeline associated with the effect instance.

Return

- *kOfxStatOK* - the time enquiry was successful
- *kOfxStatFailed* - the enquiry failed for some host specific reason
- *kOfxStatErrBadHandle* - the effect handle was invalid

OfxStatus (***gotoTime**)(void *instance, double time)

Move the timeline control to the indicated time.

- *instance* is the instance of the effect changing the timeline, cast to a void *
- *time* is the time to change the timeline to. This is in the temporal coordinate system of the effect.

This function moves the timeline to the indicated frame and returns. Any side effects of the timeline change are also triggered and completed before this returns (for example instance changed actions and renders if the output of the effect is being viewed).

Return

- *kOfxStatOK* - the time was changed successfully, will all side effects if the change completed
- *kOfxStatFailed* - the change failed for some host specific reason
- *kOfxStatErrBadHandle* - the effect handle was invalid
- *kOfxStatErrValue* - the time was an illegal value

OfxStatus (***getTimeBounds**)(void *instance, double *firstTime, double *lastTime)

Get the current bounds on a timeline.

- *instance* is the instance of the effect changing the timeline, cast to a void *
- *firstTime* is the first time on the timeline. This is in the temporal coordinate system of the effect.
- *lastTime* is last time on the timeline. This is in the temporal coordinate system of the effect.

This function

Return

- *kOfxStatOK* - the time enquiry was successful
- *kOfxStatFailed* - the enquiry failed for some host specific reason
- *kOfxStatErrBadHandle* - the effect handle was invalid

struct **OfxYUVAColourB**

#include <ofxOld.h> Defines an 8 bit per component YUVA pixel — *ofxPixels.h* Deprecated in 1.3, removed in 1.4.

Public Members

unsigned char **y**

unsigned char **u**

unsigned char **v**

unsigned char **a**

struct **OfxYUVAColourF**

#include <ofxOld.h> Defines an floating point component YUVA pixel — *ofxPixels.h*.

Deprecated:

- Deprecated in 1.3, removed in 1.4

Public Members

float **y**

float **u**

float **v**

float **a**

struct **OfxYUVAColourS**

#include <ofxOld.h> Defines an 16 bit per component YUVA pixel — *ofxPixels.h*.

Deprecated:

- Deprecated in 1.3, removed in 1.4

Public Members

unsigned short **y**

unsigned short **u**

unsigned short **v**

unsigned short **a**

file **ofxCore.h**

#include "stdafx.h" #include <limits.h> Contains the core OFX architectural struct and function definitions. For more details on the basic OFX architecture, see Architecture.

Defines

OfxExport

Platform independent export macro.

This macro is to be used before any symbol that is to be exported from a plug-in. This is OS/compiler dependent.

kOfxActionLoad

This action is the first action passed to a plug-in after the binary containing the plug-in has been loaded. It is there to allow a plug-in to create any global data structures it may need and is also when the plug-in should fetch suites from the host.

The handle, inArgs and outArgs arguments to the mainEntry are redundant and should be set to NULL.

Pre

- The plugin's *OfxPlugin::setHost* function has been called

Post

This action will not be called again while the binary containing the plug-in remains loaded.

Returns

- *kOfxStatOK*, the action was trapped and all was well,
- *kOfxStatReplyDefault*, the action was ignored,
- *kOfxStatFailed*, the load action failed, no further actions will be passed to the plug-in. Interpret if possible kOfxStatFailed as plug-in indicating it does not want to load Do not create an entry in the host's UI for plug-in then.

Plug-in also has the option to return 0 for OfxGetNumberOfPlugins or kOfxStatFailed if host supports OfxSetHost in which case kOfxActionLoad will never be called.

- *kOfxStatErrFatal*, fatal error in the plug-in.

kOfxActionDescribe

The `kOfxActionDescribe` is the second action passed to a plug-in. It is where a plugin defines how it behaves and the resources it needs to function.

Note that the handle passed in acts as a descriptor for, rather than an instance of the plugin. The handle is global and unique. The plug-in is at liberty to cache the handle away for future reference until the plug-in is unloaded.

Most importantly, the effect must set what image effect contexts it is capable of working in.

This action *must* be trapped, it is not optional.

Parameters

- **handle** – handle to the plug-in descriptor, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionLoad* has been called

Post

- *kOfxActionDescribe* will not be called again, unless it fails and returns one of the error codes where the host is allowed to attempt the action again
- the handle argument, being the global plug-in description handle, is a valid handle from the end of a successful describe action until the end of the *kOfxActionUnload* action (ie: the plug-in can cache it away without worrying about it changing between actions).
- *kOfxImageEffectActionDescribeInContext* will be called once for each context that the host and plug-in mutually support. If a plug-in does not report to support any context supported by host, host should not enable the plug-in.

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatErrMissingHostFeature*, in which the plugin will be unloaded and ignored, plugin may post message
- *kOfxStatErrMemory*, in which case describe may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxActionUnload

This action is the last action passed to the plug-in before the binary containing the plug-in is unloaded. It is there to allow a plug-in to destroy any global data structures it may have created.

The handle, inArgs and outArgs arguments to the main entry are redundant and should be set to NULL.

Pre

- the *kOfxActionLoad* action has been called
- all instances of a plugin have been destroyed

Post

- No other actions will be called.

Returns

- *kOfxStatOK*, the action was trapped all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*, in which case we the program will be forced to quit

kOfxActionPurgeCaches

This action is an action that may be passed to a plug-in instance from time to time in low memory situations. Instances receiving this action should destroy any data structures they may have and release the associated memory, they can later reconstruct this from the effect's parameter set and associated information.

For Image Effects, it is generally a bad idea to call this after each render, but rather it should be called after *kOfxImageEffectActionEndSequenceRender*. Some effects, typically those flagged with the *kOfxImageEffectInstancePropSequentialRender* property, may need to cache information from previously rendered frames to function correctly, or have data structures that are expensive to reconstruct at each frame (eg: a particle system). Ideally, such effect should free such structures during the *kOfxImageEffectActionEndSequenceRender* action.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionSyncPrivateData

This action is called when a plugin should synchronise any private data structures to its parameter set. This generally occurs when an effect is about to be saved or copied, but it could occur in other situations as well.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Post

- Any private state data can be reconstructed from the parameter set,

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionCreateInstance

This action is the first action passed to a plug-in's instance after its creation. It is there to allow a plugin to create any per-instance data structures it may need.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionDescribe* has been called
- the instance is fully constructed, with all objects requested in the describe actions (eg, parameters and clips) have been constructed and have had their initial values set. This means that if the values are being loaded from an old setup, that load should have taken place before the create instance action is called.

Post

- the instance pointer will be valid until the *kOfxActionDestroyInstance* action is passed to the plug-in with the same instance handle

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored, but all was well anyway
- *kOfxStatErrFatal*
- *kOfxStatErrMemory*, in which case this may be called again after a memory purge
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message if possible and the host should destroy the instance handle and not attempt to proceed further

kOfxActionDestroyInstance

This action is the last passed to a plug-in's instance before its destruction. It is there to allow a plugin to destroy any per-instance data structures it may have created.

- *kOfxStatOK*, the action was trapped and all was well,
- *kOfxStatReplyDefault*, the action was ignored as the effect had nothing to do,
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the handle,
- the instance has not had any of its members destroyed yet,

Post

- the instance pointer is no longer valid and any operation on it will be undefined

Returns

To some extent, what is returned is moot, a bit like throwing an exception in a C++ destructor, so the host should continue destruction of the instance regardless.

kOfxActionInstanceChanged

This action signals that something has changed in a plugin's instance, either by user action, the host or the plugin itself. All change actions are bracketed by a pair of *kOfxActionBeginInstanceChanged* and *kOfxActionEndInstanceChanged* actions. The **inArgs** property set is used to determine what was the thing inside the instance that was changed.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropType* The type of the thing that changed which will be one of..
 - *kOfxTypeParameter* Indicating a parameter's value has changed in some way
 - *kOfxTypeClip* A clip to an image effect has changed in some way (for Image Effect Plugins only)
 - *kOfxPropName* the name of the thing that was changed in the instance
 - *kOfxPropChangeReason* what triggered the change, which will be one of...
 - *kOfxChangeUserEdited* - the user or host changed the instance somehow and caused a change to something, this includes undo/redos, resets and loading values from files or presets,
 - *kOfxChangePluginEdited* - the plugin itself has changed the value of the instance in some action
 - *kOfxChangeTime* - the time has changed and this has affected the value of the object because it varies over time
 - *kOfxPropTime*
 - the effect time at which the change occurred (for Image Effect Plugins only)
 - *kOfxImageEffectPropRenderScale*
 - the render scale currently being applied to any image fetched from a clip (for Image Effect Plugins only)
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- *kOfxActionBeginInstanceChanged* has been called on the instance handle.

Post

- *kOfxActionEndInstanceChanged* will be called on the instance handle.

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionBeginInstanceChanged

The *kOfxActionBeginInstanceChanged* and *kOfxActionEndInstanceChanged* actions are used to bracket all *kOfxActionInstanceChanged* actions, whether a single change or multiple changes. Some changes to a plugin instance can be grouped logically (eg: a ‘reset all’ button resetting all the instance’s parameters), the begin/end instance changed actions allow a plugin to respond appropriately to a large set of changes. For example, a plugin that maintains a complex internal state can delay any changes to that state until all parameter changes have completed.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropChangeReason* what triggered the change, which will be one of...
 - *kOfxChangeUserEdited* - the user or host changed the instance somehow and caused a change to something, this includes undo/redos, resets and loading values from files or presets,
 - *kOfxChangePluginEdited* - the plugin itself has changed the value of the instance in some action
 - *kOfxChangeTime* - the time has changed and this has affected the value of the object because it varies over time
- **outArgs** – is redundant and is set to NULL

Post

- For *kOfxActionBeginInstanceChanged* , *kOfxActionCreateInstance* has been called on the instance handle.
- For *kOfxActionEndInstanceChanged* , *kOfxActionBeginInstanceChanged* has been called on the instance handle.
- *kOfxActionCreateInstance* has been called on the instance handle.

Post

- For *kOfxActionBeginInstanceChanged*, *kOfxActionInstanceChanged* will be called at least once on the instance handle.
- *kOfxActionEndInstanceChanged* will be called on the instance handle.

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionEndInstanceChanged

Action called after the end of a set of *kOfxActionEndInstanceChanged* actions, used with *kOfxActionBeginInstanceChanged* to bracket a grouped set of changes, see *kOfxActionBeginInstanceChanged*.

kOfxActionBeginInstanceEdit

This is called when an instance is *first* actively edited by a user, ie: and interface is open and parameter values and input clips can be modified. It is there so that effects can create private user interface structures when necessary. Note that some hosts can have multiple editors open on the same effect instance simultaneously.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Post

- *kOfxActionEndInstanceEdit* will be called when the last editor is closed on the instance

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxActionEndInstanceEdit

This is called when the *last* user interface on an instance closed. It is there so that effects can destroy private user interface structures when necessary. Note that some hosts can have multiple editors open on the same effect instance simultaneously, this will only be called when the last of those editors are closed.

Parameters

- **handle** – handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionBeginInstanceEdit* has been called on the instance handle,

Post

- no user interface is open on the instance

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored
- *kOfxStatErrFatal*,
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plugin should to post a message

kOfxPropAPIVersion

Property on the host descriptor, saying what API version of the API is being implemented.

- Type - int X N
- Property Set - host descriptor.

This is a version string that will specify which version of the API is being implemented by a host. It can have multiple values. For example “1.0”, “1.2.4” etc....

If this is not present, it is safe to assume that the version of the API is “1.0”.

kOfxPropTime

General property used to get/set the time of something.

- Type - double X 1
- Default - 0, if a settable property
- Property Set - commonly used as an argument to actions, input and output.

kOfxPropIsInteractive

Indicates if a host is actively editing the effect with some GUI.

- Type - int X 1
- Property Set - effect instance (read only)
- Valid Values - 0 or 1

If false the effect currently has no interface, however this may be because the effect is loaded in a background render host, or it may be loaded on an interactive host that has not yet opened an editor for the effect.

The output of an effect should only ever depend on the state of its parameters, not on the interactive flag. The interactive flag is more a courtesy flag to let a plugin know that it has an interace. If a plugin want's to have its behaviour dependant on the interactive flag, it can always make a secret parameter which shadows the state if the flag.

kOfxPluginPropFilePath

The file path to the plugin.

- Type - C string X 1
- Property Set - effect descriptor (read only)

This is a string that indicates the file path where the plug-in was found by the host. The path is in the native path format for the host OS (eg: UNIX directory separators are forward slashes, Windows ones are backslashes).

The path is to the bundle location, see `InstallationLocation`. eg:
 ‘/usr/OFX/Plugins/AcmePlugins/AcmeFantasticPlugin.ofx.bundle’

kOfxPropInstanceData

A private data pointer that the plug-in can store its own data behind.

- Type - pointer X 1
- Property Set - plugin instance (read/write),
- Default - NULL

This data pointer is unique to each plug-in instance, so two instances of the same plug-in do not share the same data pointer. Use it to hang any needed private data structures.

kOfxPropType

General property, used to identify the kind of an object behind a handle.

- Type - ASCII C string X 1
- Property Set - any object handle (read only)
- Valid Values - currently this can be...
 - *kOfxTypeImageEffectHost*
 - *kOfxTypeImageEffect*
 - *kOfxTypeImageEffectInstance*
 - *kOfxTypeParameter*
 - *kOfxTypeParameterInstance*
 - *kOfxTypeClip*
 - *kOfxTypeImage*

kOfxPropName

Unique name of an object.

- Type - ASCII C string X 1
- Property Set - on many objects (descriptors and instances), see `PropertiesByObject` (read only)

This property is used to label objects uniquely among objects of that type. It is typically set when a plugin creates a new object with a function that takes a name.

kOfxPropVersion

Identifies a specific version of a host or plugin.

- Type - int X N
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - “0”
- Valid Values - positive integers

This is a multi dimensional integer property that represents the version of a host (host descriptor), or plugin (plugin descriptor). These represent a version number of the form ‘1.2.3.4’, with each dimension adding another ‘dot’ on the right.

A version is considered to be more recent than another if its ordered set of values is lexicographically greater than another, reading left to right. (ie: 1.2.4 is smaller than 1.2.6). Also, if the number of dimensions is different, then the values of the missing dimensions are considered to be zero (so 1.2.4 is greater than 1.2).

kOfxPropVersionLabel

Unique user readable version string of a plugin or host.

- Type - string X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - none, the host needs to set this
- Valid Values - ASCII string

This is purely for user feedback, a plugin or host should use *kOfxPropVersion* if they need to check for specific versions.

kOfxPropPluginDescription

Description of the plug-in to a user.

- Type - string X 1
- Property Set - plugin descriptor (read/write) and instance (read only)
- Default - “”
- Valid Values - UTF8 string

This is a string giving a potentially verbose description of the effect.

kOfxPropLabel

User visible name of an object.

- Type - UTF8 C string X 1
- Property Set - on many objects (descriptors and instances), see PropertiesByObject. Typically readable and writable in most cases.
- Default - the *kOfxPropName* the object was created with.

The label is what a user sees on any interface in place of the object's name.

Note that resetting this will also reset *kOfxPropShortLabel* and *kOfxPropLongLabel*.

kOfxPropIcon

If set this tells the host to use an icon instead of a label for some object in the interface.

- Type - string X 2
- Property Set - various descriptors in the API
- Default - ""
- Valid Values - ASCII string

The value is a path is defined relative to the Resource folder that points to an SVG or PNG file containing the icon.

The first dimension, if set, will the name of and SVG file, the second a PNG file.

kOfxPropShortLabel

Short user visible name of an object.

- Type - UTF8 C string X 1
- Property Set - on many objects (descriptors and instances), see *PropertiesByObject*. Typically readable and writable in most cases.
- Default - initially *kOfxPropName*, but will be reset if *kOfxPropLabel* is changed.

This is a shorter version of the label, typically 13 character glyphs or less. Hosts should use this if they have limited display space for their object labels.

kOfxPropLongLabel

Long user visible name of an object.

- Type - UTF8 C string X 1
- Property Set - on many objects (descriptors and instances), see *PropertiesByObject*. Typically readable and writable in most cases.
- Default - initially *kOfxPropName*, but will be reset if *kOfxPropLabel* is changed.

This is a longer version of the label, typically 32 character glyphs or so. Hosts should use this if they have much display space for their object labels.

kOfxPropChangeReason

Indicates why a plug-in changed.

- Type - ASCII C string X 1
- Property Set - inArgs parameter on the *kOfxActionInstanceChanged* action.
- Valid Values - this can be...

- *kOfxChangeUserEdited* - the user directly edited the instance somehow and caused a change to something, this includes undo/redos and resets
- *kOfxChangePluginEdited* - the plug-in itself has changed the value of the object in some action
- *kOfxChangeTime* - the time has changed and this has affected the value of the object because it varies over time

Argument property for the *kOfxActionInstanceChanged* action.

kOfxPropEffectInstance

A pointer to an effect instance.

- Type - pointer X 1
- Property Set - on an interact instance (read only)

This property is used to link an object to the effect. For example if the plug-in supplies an OpenGL overlay for an image effect, the interact instance will have one of these so that the plug-in can connect back to the effect the GUI links to.

kOfxPropHostOSHandle

A pointer to an operating system specific application handle.

- Type - pointer X 1
- Property Set - host descriptor.

Some plug-in vendor want raw OS specific handles back from the host so they can do interesting things with host OS APIs. Typically this is to control windowing properly on Microsoft Windows. This property returns the appropriate 'root' window handle on the current operating system. So on Windows this would be the hWnd of the application main window.

kOfxChangeUserEdited

String used as a value to *kOfxPropChangeReason* to indicate a user has changed something.

kOfxChangePluginEdited

String used as a value to *kOfxPropChangeReason* to indicate the plug-in itself has changed something.

kOfxChangeTime

String used as a value to *kOfxPropChangeReason* to a time varying object has changed due to a time change.

kOfxFlagInfiniteMax

Used to flag infinite rects. Set minimums to this to indicate infinite.

This is effectively INT_MAX.

kOfxFlagInfiniteMin

Used to flag infinite rects. Set minimums to this to indicate infinite.

This is effectively INT_MIN

kOfxBitDepthNone

String used to label unset bitdepths.

kOfxBitDepthByte

String used to label unsigned 8 bit integer samples.

kOfxBitDepthShort

String used to label unsigned 16 bit integer samples.

kOfxBitDepthHalf

String used to label half-float (16 bit floating point) samples.

Version

Added in Version 1.4. Was in *ofxOpenGLRender.h* before.

kOfxBitDepthFloat

String used to label signed 32 bit floating point samples.

kOfxStatOK

Status code indicating all was fine.

kOfxStatFailed

Status error code for a failed operation.

kOfxStatErrFatal

Status error code for a fatal error.

Only returned in the case where the plug-in or host cannot continue to function and needs to be restarted.

kOfxStatErrUnknown

Status error code for an operation on or request for an unknown object.

kOfxStatErrMissingHostFeature

Status error code returned by plug-ins when they are missing host functionality, either an API or some optional functionality (eg: custom params).

Plug-Ins returning this should post an appropriate error message stating what they are missing.

kOfxStatErrUnsupported

Status error code for an unsupported feature/operation.

kOfxStatErrExists

Status error code for an operation attempting to create something that exists.

kOfxStatErrFormat

Status error code for an incorrect format.

kOfxStatErrMemory

Status error code indicating that something failed due to memory shortage.

kOfxStatErrBadHandle

Status error code for an operation on a bad handle.

kOfxStatErrBadIndex

Status error code indicating that a given index was invalid or unavailable.

kOfxStatErrValue

Status error code indicating that something failed due an illegal value.

kOfxStatReplyYes

OfxStatus returned indicating a 'yes'.

kOfxStatReplyNo

OfxStatus returned indicating a 'no'.

kOfxStatReplyDefault

OfxStatus returned indicating that a default action should be performed.

Typedefs

typedef struct OfxPropertySetStruct ***OfxPropertySetHandle**

Blind data structure to manipulate sets of properties through.

typedef int **OfxStatus**

OFX status return type.

typedef struct *OfxHost* **OfxHost**

Generic host structure passed to *OfxPlugin::setHost* function.

This structure contains what is needed by a plug-in to bootstrap its connection to the host.

OfxStatus() **OfxPluginEntryPoint** (**const char *action, const void *handle, OfxPropertySetHandle inArgs, OfxPropertySetHandle outArgs**)

Entry point for plug-ins.

- *action* ASCII c string indicating which action to take
- *instance* object to which action should be applied, this will need to be cast to the appropriate blind data type depending on the *action*
- *inData* handle that contains action specific properties
- *outData* handle where the plug-in should set various action specific properties

This is how the host generally communicates with a plug-in. Entry points are used to pass messages to various objects used within OFX. The main use is within the *OfxPlugin* struct.

The exact set of actions is determined by the plug-in API that is being implemented, however all plug-ins can perform several actions. For the list of actions consult OFX Actions.

typedef struct *OfxPlugin* **OfxPlugin**

The structure that defines a plug-in to a host.

This structure is the first element in any plug-in structure using the OFX plug-in architecture. By examining its members a host can determine the API that the plug-in implements, the version of that API, its name and version.

For details see Architecture.

typedef double **OfxTime**

How time is specified within the OFX API.

typedef struct *OfxRangeI* **OfxRangeI**

Defines one dimensional integer bounds.

typedef struct *OfxRangeD* **OfxRangeD**

Defines one dimensional double bounds.

typedef struct *OfxPointI* **OfxPointI**

Defines two dimensional integer point.

typedef struct *OfxPointD* **OfxPointD**

Defines two dimensional double point.

typedef struct *OfxRectI* **OfxRectI**

Defines two dimensional integer region.

Regions are $x1 \leq x < x2$

Infinite regions are flagged by setting

- $x1 = kOfxFlagInfiniteMin$
- $y1 = kOfxFlagInfiniteMin$
- $x2 = kOfxFlagInfiniteMax$
- $y2 = kOfxFlagInfiniteMax$

typedef struct *OfxRectD* **OfxRectD**

Defines two dimensional double region.

Regions are $x1 \leq x < x2$

Infinite regions are flagged by setting

- $x1 = kOfxFlagInfiniteMin$
- $y1 = kOfxFlagInfiniteMin$
- $x2 = kOfxFlagInfiniteMax$

- $y2 = kOfxFlagInfiniteMax$

Functions

OfxPlugin *OfxGetPlugin(int nth)

Returns the 'nth' plug-in implemented inside a binary.

Returns a pointer to the 'nth' plug-in implemented in the binary. A function of this type must be implemented in and exported from each plug-in binary.

int OfxGetNumberOfPlugins(void)

Defines the number of plug-ins implemented inside a binary.

A host calls this to determine how many plug-ins there are inside a binary it has loaded. A function of this type must be implemented in and exported from each plug-in binary.

OfxStatus OfxSetHost(const *OfxHost* *host)

First thing host should call.

This host call, added in 2020, is not specified in earlier implementation of the API. Therefore host must check if the plugin implemented it and not assume symbol exists. The order of calls is then: 1) OfxSetHost, 2) OfxGetNumberOfPlugins, 3) OfxGetPlugin The host pointer is only assumed valid until OfxGetPlugin where it might get reset. Plug-in can return kOfxStatFailed to indicate it has nothing to do here, it's not for this Host and it should be skipped silently.

file ofxDialo.h

#include "ofxCore.h" #include "ofxProperty.h" This file contains an optional suite which should be used to popup a native OS dialog from a host parameter changed action.

When a host uses a fullscreen window and is running the OFX plugins in another thread it can lead to a lot of conflicts if that plugin will try to open its own window.

This suite will provide the functionality for a plugin to request running its dialog in the UI thread, and informing the host it will do this so it can take the appropriate actions needed. (Like lowering its priority etc..)

Defines

kOfxDialoSuite

The name of the Dialog suite, used to fetch from a host via *OfxHost::fetchSuite*.

kOfxActionDialog

Action called after a dialog has requested a 'Dialog' The arguments to the action are:

- *user_data* Pointer which was provided when the plugin requested the Dialog

When the plugin receives this action it is safe to popup a dialog. It runs in the host's UI thread, which may differ from the main OFX processing thread. Plugin should return from this action when all Dialog interactions are done. At that point the host will continue again. The host will not send any other messages asynchronous to this one.

Typedefs

typedef struct *OfxDialogSuiteV1* **OfxDialogSuiteV1**

file **ofxDrawSuite.h**

#include "ofxCore.h" #include "ofxPixels.h" API for host- and GPU API-independent drawing.

Version

Added in OpenFX 1.5

Defines

kOfxDrawSuite

the string that names the DrawSuite, passed to *OfxHost::fetchSuite*

kOfxInteractPropDrawContext

The Draw Context handle.

- Type - pointer X 1
- Property Set - read only property on the inArgs of the following actions...
- *kOfxInteractActionDraw*

Typedefs

typedef struct OfxDrawContext ***OfxDrawContextHandle**

Blind declaration of an OFX drawing context.

typedef enum *OfxStandardColour* **OfxStandardColour**

Defines valid values for *OfxDrawSuiteV1::getColour*.

typedef enum *OfxDrawLineStipplePattern* **OfxDrawLineStipplePattern**

Defines valid values for *OfxDrawSuiteV1::setLineStipple*.

typedef enum *OfxDrawPrimitive* **OfxDrawPrimitive**

Defines valid values for *OfxDrawSuiteV1::draw*.

typedef enum *OfxDrawTextAlignment* **OfxDrawTextAlignment**

Defines text alignment values for *OfxDrawSuiteV1::drawText*.

typedef struct *OfxDrawSuiteV1* **OfxDrawSuiteV1**

OFX suite that allows an effect to draw to a host-defined display context.

Enums

enum **OfxStandardColour**

Defines valid values for *OfxDrawSuiteV1::getColour*.

Values:

enumerator **kOfxStandardColourOverlayBackground**

enumerator **kOfxStandardColourOverlayActive**

enumerator **kOfxStandardColourOverlaySelected**

enumerator **kOfxStandardColourOverlayDeselected**

enumerator **kOfxStandardColourOverlayMarqueeFG**

enumerator **kOfxStandardColourOverlayMarqueeBG**

enumerator **kOfxStandardColourOverlayText**

enum **OfxDrawLineStipplePattern**

Defines valid values for *OfxDrawSuiteV1::setLineStipple*.

Values:

enumerator **kOfxDrawLineStipplePatternSolid**

enumerator **kOfxDrawLineStipplePatternDot**

enumerator **kOfxDrawLineStipplePatternDash**

enumerator **kOfxDrawLineStipplePatternAltDash**

enumerator **kOfxDrawLineStipplePatternDotDash**

enum **OfxDrawPrimitive**

Defines valid values for *OfxDrawSuiteV1::draw*.

Values:

enumerator **kOfxDrawPrimitiveLines**

enumerator **kOfxDrawPrimitiveLineStrip**

enumerator **kOfxDrawPrimitiveLineLoop**

enumerator **kOfxDrawPrimitiveRectangle**

enumerator **kOfxDrawPrimitivePolygon**

enumerator **kOfxDrawPrimitiveEllipse**

enum **OfxDrawTextAlignment**

Defines text alignment values for *OfxDrawSuiteV1::drawText*.

Values:

enumerator **kOfxDrawTextAlignmentLeft**

enumerator **kOfxDrawTextAlignmentRight**

enumerator **kOfxDrawTextAlignmentTop**

enumerator **kOfxDrawTextAlignmentBottom**

enumerator **kOfxDrawTextAlignmentBaseline**

enumerator **kOfxDrawTextAlignmentCenterH**

enumerator **kOfxDrawTextAlignmentCenterV**

file **ofxGPURender.h**

`#include "ofxImageEffect.h"`

This file contains an optional suite **for** performing GPU-accelerated rendering of OpenFX Image Effect Plug-ins. For details see [\ref ofxGPURender](#).

It allows hosts **and** plug-ins to support OpenGL, OpenCL, CUDA, **and** Metal. Additional GPU APIs, such as Vulkan, could use similar techniques.

StatusReturnValues

OfxStatus returns indicating that a OpenGL render error has occurred:

- If a plug-in returns *kOfxStatGLRenderFailed*, the host should retry the render with OpenGL rendering disabled.
- If a plug-in returns *kOfxStatGLOutOfMemory*, the host may choose to free resources on the GPU and retry the OpenGL render, rather than immediately falling back to CPU rendering.

kOfxStatGPUOutOfMemory

GPU render ran out of memory.

kOfxStatGLOutOfMemory

OpenGL render ran out of memory (same as kOfxStatGPUOutOfMemory)

kOfxStatGPURenderFailed

GPU render failed in a non-memory-related way.

kOfxStatGLRenderFailed

OpenGL render failed in a non-memory-related way (same as kOfxStatGPURenderFailed)

Defines

`__OFXGPURENDER_H__`

kOfxOpenGLRenderSuite

The name of the OpenGL render suite, used to fetch from a host via *OfxHost::fetchSuite*.

kOfxImageEffectPropOpenGLRenderSupported

Indicates whether a host or plug-in can support OpenGL accelerated rendering.

- Type - C string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only) - plug-in instance change (read/write)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - in which case the host or plug-in does not support OpenGL accelerated rendering
 - “true” - which means a host or plug-in can support OpenGL accelerated rendering, in the case of plug-ins this also means that it is capable of CPU based rendering in the absence of a GPU
 - “needed” - only for plug-ins, this means that an plug-in has to have OpenGL support, without which it cannot work.

V1.4: It is now expected from host reporting v1.4 that the plug-in can during instance change switch from true to false and false to true.

kOfxOpenGLPropPixelDepth

Indicates the bit depths supported by a plug-in during OpenGL renders.

This is analogous to *kOfxImageEffectPropSupportedPixelDepths*. When a plug-in sets this property, the host will try to provide buffers/textures in one of the supported formats. Additionally, the target buffers where the plug-in renders to will be set to one of the supported formats.

Unlike *kOfxImageEffectPropSupportedPixelDepths*, this property is optional. Shader-based effects might not really care about any format specifics when using OpenGL textures, so they can leave this unset and allow the host the decide the format.

- Type - string X N
- Property Set - plug-in descriptor (read only)
- Default - none set
- Valid Values - This must be one of
 - *kOfxBitDepthNone* (implying a clip is unconnected, not valid for an image)
 - *kOfxBitDepthByte*
 - *kOfxBitDepthShort*
 - *kOfxBitDepthHalf*
 - *kOfxBitDepthFloat*

kOfxImageEffectPropOpenGLEnabled

Indicates that a plug-in SHOULD use OpenGL acceleration in the current action.

When a plug-in and host have established they can both use OpenGL renders then when this property has been set the host expects the plug-in to render its result into the buffer it has setup before calling the render. The plug-in can then also safely use the 'OfxImageEffectOpenGLRenderSuite'

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the plug-in cannot use the OpenGL suite
 - 1 indicates that the plug-in should render into the texture, and may use the OpenGL suite functions.

v1.4: kOfxImageEffectPropOpenGLEnabled should probably be checked in Instance Changed prior to try to read image via clipLoadTexture

Note: Once this property is set, the host and plug-in have agreed to use OpenGL, so the effect SHOULD access all its images through the OpenGL suite.

kOfxImageEffectPropOpenGLTextureIndex

Indicates the texture index of an image turned into an OpenGL texture by the host.

- Type - int X 1
- Property Set - texture handle returned by `OfxImageEffectOpenGLRenderSuiteV1::clipLoadTexture` (read only)

This value should be cast to a GLuint **and** used **as** the texture index when performing OpenGL texture operations.

The property set of the following actions should contain this property:

- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*

kOfxImageEffectPropOpenGLTextureTarget

Indicates the texture target enumerator of an image turned into an OpenGL texture by the host.

- Type - int X 1
- Property Set - texture handle returned by *OfxImageEffectOpenGLRenderSuiteV1::clipLoadTexture* (read only) This value should be cast to a GLenum and used as the texture target when performing OpenGL texture operations.

The property set of the following actions should contain this property:

- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*

kOfxActionOpenGLContextAttached

Action called when an effect has just been attached to an OpenGL context.

The purpose of this action is to allow a plug-in to set up any data it may need to do OpenGL rendering in an instance. For example...

- allocate a lookup table on a GPU,
- create an OpenCL or CUDA context that is bound to the host's OpenGL context so it can share buffers.

The plug-in will be responsible for deallocating any such shared resource in the *kOfxActionOpenGLContextDetached* action.

A host cannot call *kOfxActionOpenGLContextAttached* on the same instance without an intervening *kOfxActionOpenGLContextDetached*. A host can have a plug-in swap OpenGL contexts by issuing a attach/detach for the first context then another attach for the next context.

The arguments to the action are...

- *handle* handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- *inArgs* is redundant and set to NULL
- *outArgs* is redundant and set to NULL

A plug-in can return...

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored, but all was well anyway
- *kOfxStatErrMemory*, in which case this may be called again after a memory purge

- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plug-in should to post a message if possible and the host should not attempt to run the plug-in in OpenGL render mode.

kOfxActionOpenGLContextDetached

Action called when an effect is about to be detached from an OpenGL context.

The purpose of this action is to allow a plug-in to deallocate any resource allocated in *kOfxActionOpenGLContextAttached* just before the host decouples a plug-in from an OpenGL context. The host must call this with the same OpenGL context active as it called with the corresponding *kOfxActionOpenGLContextAttached*.

The arguments to the action are...

- *handle* handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- *inArgs* is redundant and set to NULL
- *outArgs* is redundant and set to NULL

A plug-in can return...

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored, but all was well anyway
- *kOfxStatErrMemory*, in which case this may be called again after a memory purge
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plug-in should to post a message if possible and the host should not attempt to run the plug-in in OpenGL render mode.

kOfxImageEffectPropCudaRenderSupported

Indicates whether a host or plug-in can support CUDA render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - "false" for a plug-in
- Valid Values - This must be one of
 - "false" - the host or plug-in does not support CUDA render
 - "true" - the host or plug-in can support CUDA render

kOfxImageEffectPropCudaEnabled

Indicates that a plug-in SHOULD use CUDA render in the current action.

If a plug-in and host have both set *kOfxImageEffectPropCudaRenderSupported*="true" then the host MAY set this property to indicate that it is passing images as CUDA memory pointers.

- Type - int X 1
- Property Set - *inArgs* property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

- Valid Values
 - 0 indicates that the `kOfxImagePropData` of each image of each clip is a CPU memory pointer.
 - 1 indicates that the `kOfxImagePropData` of each image of each clip is a CUDA memory pointer.

`kOfxImageEffectPropCudaStreamSupported`

Indicates whether a host or plug-in can support CUDA streams.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - in which case the host or plug-in does not support CUDA streams
 - “true” - which means a host or plug-in can support CUDA streams

`kOfxImageEffectPropCudaStream`

The CUDA stream to be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *`kOfxImageEffectActionRender`*
 - *`kOfxImageEffectActionBeginSequenceRender`*
 - *`kOfxImageEffectActionEndSequenceRender`*

This property will only be set if the host and plug-in both support CUDA streams.

If set:

- this property contains a pointer to the stream of CUDA render (`cudaStream_t`). In order to use it, `reinterpret_cast<cudaStream_t>(pointer)` is needed.
- the plug-in SHOULD ensure that its render action enqueues any asynchronous CUDA operations onto the supplied queue.
- the plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action, and SHOULD NOT call `cudaDeviceSynchronize()` at any time.

If not set:

- the plug-in SHOULD ensure that any asynchronous operations it enqueues have completed before returning from the render action.

`kOfxImageEffectPropMetalRenderSupported`

Indicates whether a host or plug-in can support Metal render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support Metal render
 - “true” - the host or plug-in can support Metal render

kOfxImageEffectPropMetalEnabled

Indicates that a plug-in SHOULD use Metal render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropMetalRenderSupported=“true”` then the host MAY set this property to indicate that it is passing images as Metal buffers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the `kOfxImagePropData` of each image of each clip is a CPU memory pointer.
 - 1 indicates that the `kOfxImagePropData` of each image of each clip is a Metal id<MTLBuffer>.

kOfxImageEffectPropMetalCommandQueue

The command queue of Metal render.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property contains a pointer to the command queue to be used for Metal rendering (id<MTLCommandQueue>). In order to use it, `reinterpret_cast<id<MTLCommandQueue>>(pointer)` is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous Metal operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

kOfxImageEffectPropOpenCLRenderSupported

Indicates whether a host or plug-in can support OpenCL Buffers render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support OpenCL Buffers render
 - “true” - the host or plug-in can support OpenCL Buffers render

kOfxImageEffectPropOpenCLSupported

Indicates whether a host or plug-in can support OpenCL Images render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - in which case the host or plug-in does not support OpenCL Images render
 - “true” - which means a host or plug-in can support OpenCL Images render

kOfxImageEffectPropOpenCLEnabled

Indicates that a plug-in SHOULD use OpenCL render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropOpenCLRenderSupported=“true”` or have both set `kOfxImageEffectPropOpenCLSupported=“true”` then the host MAY set this property to indicate that it is passing images as OpenCL Buffers or Images.

When rendering using OpenCL Buffers, the `cl_mem` of the buffers are retrieved using *kOfxImagePropData*. When rendering using OpenCL Images, the `cl_mem` of the images are retrieved using *kOfxImageEffectPropOpenCLImage*. If both *kOfxImageEffectPropOpenCLSupported* (Buffers) and *kOfxImageEffectPropOpenCLRenderSupported* (Images) are enabled by the plug-in, it should use *kOfxImageEffectPropOpenCLImage* to determine which is being used by the host.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that a plug-in SHOULD use OpenCL render in the render action
 - 1 indicates that a plug-in SHOULD NOT use OpenCL render in the render action

kOfxImageEffectPropOpenCLCommandQueue

Indicates the OpenCL command queue that should be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property contains a pointer to the command queue to be used for OpenCL rendering (`cl_command_queue`). In order to use it, `reinterpret_cast<cl_command_queue>(pointer)` is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous OpenCL operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

kOfxImageEffectPropOpenCLImage

Indicates the image handle of an image supplied as an OpenCL Image by the host.

- Type - pointer X 1
- Property Set - image handle returned by `clipGetImage`

This value should be cast to a `cl_mem` and used as the image handle when performing OpenCL Images operations. The property should be used (not *kOfxImagePropData*) when rendering with OpenCL Images (*kOfxImageEffectPropOpenCLSupported*), and should be used to determine whether Images or Buffers should be used if a plug-in supports both *kOfxImageEffectPropOpenCLSupported* and *kOfxImageEffectPropOpenCLRenderSupported*. Note: the `kOfxImagePropRowBytes` property is not required to be set by the host, since OpenCL Images do not have the concept of row bytes.

kOfxOpenCLProgramSuite

Typedefs

```
typedef struct OfxImageEffectOpenGLRenderSuiteV1 OfxImageEffectOpenGLRenderSuiteV1
```

OFX suite that provides image to texture conversion for OpenGL processing.

```
typedef struct OfxOpenCLProgramSuiteV1 OfxOpenCLProgramSuiteV1
```

OFX suite that allows a plug-in to get OpenCL programs compiled.

This is an optional suite the host can provide for building OpenCL programs for the plug-in, as an alternative to calling `clCreateProgramWithSource / clBuildProgram`. There are two advantages to doing this: The host can add flags (such as `-cl-denorms-are-zero`) to the build call, and may also cache program binaries for performance (however, if the source of the program or the OpenCL environment changes, the host must recompile so some mechanism such as hashing must be used).

file **ofxImageEffect.h**

```
#include "ofxCore.h"#include "ofxParam.h"#include "ofxInteract.h"#include "ofxMessage.h"#include "ofxMemory.h"#include "ofxMultiThread.h"
```

Defines

`_ofxImageEffect_h_`

`kOfxImageEffectPluginApi`

String used to label OFX Image Effect Plug-ins.

Set the `pluginApi` member of the `OfxPluginHeader` inside any `OfxImageEffectPluginStruct` to be this so that the host knows the plugin is an image effect.

`kOfxImageEffectPluginApiVersion`

The current version of the Image Effect API.

`kOfxImageComponentNone`

String to label something with unset components.

`kOfxImageComponentRGBA`

String to label images with RGBA components.

`kOfxImageComponentRGB`

String to label images with RGB components.

`kOfxImageComponentAlpha`

String to label images with only Alpha components.

`kOfxImageEffectContextGenerator`

Use to define the generator image effect context. See *`kOfxImageEffectPropContext`*.

`kOfxImageEffectContextFilter`

Use to define the filter effect image effect context See *`kOfxImageEffectPropContext`*.

`kOfxImageEffectContextTransition`

Use to define the transition image effect context See *`kOfxImageEffectPropContext`*.

`kOfxImageEffectContextPaint`

Use to define the paint image effect context See *`kOfxImageEffectPropContext`*.

`kOfxImageEffectContextGeneral`

Use to define the general image effect context See *`kOfxImageEffectPropContext`*.

`kOfxImageEffectContextRetimer`

Use to define the retimer effect context See *`kOfxImageEffectPropContext`*.

`kOfxTypeImageEffectHost`

Used as a value for *`kOfxPropType`* on image effect host handles.

kOfxTypeImageEffect

Used as a value for *kOfxPropType* on image effect plugin handles.

kOfxTypeImageEffectInstance

Used as a value for *kOfxPropType* on image effect instance handles

kOfxTypeClip

Used as a value for *kOfxPropType* on image effect clips.

kOfxTypeImage

Used as a value for *kOfxPropType* on image effect images.

kOfxImageEffectActionGetRegionOfDefinition

The region of definition for an image effect is the rectangular section of the 2D image plane that it is capable of filling, given the state of its input clips and parameters. This action is used to calculate the RoD for a plugin instance at a given frame. For more details on regions of definition see Image Effect Architectures.

Note that hosts that have constant sized imagery need not call this action, only hosts that allow image sizes to vary need call this.

If the effect did not trap this, it means the host should use the default RoD instead, which depends on the context. This is...

- generator context - defaults to the project window,
- filter and paint contexts - defaults to the RoD of the ‘Source’ input clip at the given time,
- transition context - defaults to the union of the RoDs of the ‘SourceFrom’ and ‘SourceTo’ input clips at the given time,
- general context - defaults to the union of the RoDs of all the non optional input clips and the ‘Source’ input clip (if it exists and it is connected) at the given time, if none exist, then it is the project window
- retimer context - defaults to the union of the RoD of the ‘Source’ input clip at the frame directly preceding the value of the ‘SourceTime’ double parameter and the frame directly after it

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the effect time for which a region of definition is being requested
 - *kOfxImageEffectPropRenderScale* the render scale that should be used in any calculations in this action
- **outArgs** – has the following property which the plug-in may set
 - *kOfxImageEffectPropRegionOfDefinition* the calculated region of definition, initially set by the host to the default RoD (see below), in Canonical Coordinates.

Returns

- *kOfxStatOK* the action was trapped and the RoD was set in the outArgs property set

- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetRegionsOfInterest

This action allows a host to ask an effect, given a region I want to render, what region do you need from each of your input clips. In that way, depending on the host architecture, a host can fetch the minimal amount of the image needed as input. Note that there is a region of interest to be set in `outArgs` for each input clip that exists on the effect. For more details see *Image Effect Architectures*”.

The default RoI is simply the value passed in on the *kOfxImageEffectPropRegionOfInterest* `inArgs` property set. All the RoIs in the `outArgs` property set must be initialised to this value before the action is called.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the effect time for which a region of definition is being requested
 - *kOfxImageEffectPropRenderScale* the render scale that should be used in any calculations in this action
 - *kOfxImageEffectPropRegionOfInterest* the region to be rendered in the output image, in Canonical Coordinates.
- **outArgs** – has a set of 4 dimensional double properties, one for each of the input clips to the effect. The properties are each named `OfxImageClipPropRoI_` with the clip name post pended, for example `OfxImageClipPropRoI_Source`. These are initialised to the default RoI.

Returns

- *kOfxStatOK*, the action was trapped and at least one RoI was set in the `outArgs` property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetTimeDomain

This action allows a host to ask an effect what range of frames it can produce images over. Only effects instantiated in the *GeneralContext*” can have this called on them. In all other the host is in strict control over the temporal duration of the effect.

The default is:

- the union of all the frame ranges of the non optional input clips,
- infinite if there are no non optional input clips.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is null
- **outArgs** – has the following property
 - *kOfxImageEffectPropFrameRange* the frame range an effect can produce images for

Pre

- *kOfxActionCreateInstance* has been called on the instance
- the effect instance has been created in the general effect context

Returns

- *kOfxStatOK*, the action was trapped and the *kOfxImageEffectPropFrameRange* was set in the outArgs property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default value
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetFramesNeeded

This action lets the host ask the effect what frames are needed from each input clip to process a given frame. For example a temporal based degrainer may need several frames around the frame to render to do its work.

This action need only ever be called if the plugin has set the *kOfxImageEffectPropTemporalClipAccess* property on the plugin descriptor to be true. Otherwise the host assumes that the only frame needed from the inputs is the current one and this action is not called.

Note that each clip can have it's required frame range specified, and that you can specify discontinuous sets of ranges for each clip, for example

```
// The effect always needs the initial frame of the source as well as the
↳previous and current frame
double rangeSource[4];

// required ranges on the source
rangeSource[0] = 0; // we always need frame 0 of the source
rangeSource[1] = 0;
rangeSource[2] = currentFrame - 1; // we also need the previous and current
↳frame on the source
rangeSource[3] = currentFrame;

gPropHost->propSetDoubleN(outArgs, "OfxImageClipPropFrameRange_Source", 4,
↳rangeSource);
```

Which sets two discontinuous range of frames from the 'Source' clip required as input.

The default frame range is simply the single frame, *kOfxPropTime.kOfxPropTime*, found on the *inArgs* property set. All the frame ranges in the *outArgs* property set must initialised to this value before the action is called.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following property
 - *kOfxPropTime* the effect time for which we need to calculate the frames needed on input
 - **outArgs** has a set of properties, one for each input clip, named *OfxImageClipPropFrameRange_* with the name of the clip post-pended. For example *OfxImageClipPropFrameRange_Source*. All these properties are multi-dimensional doubles, with the dimension is a multiple of two. Each pair of values indicates a continuous range of frames that is needed on the given input. They are all initialised to the default value.

Returns

- *kOfxStatOK*, the action was trapped and at least one frame range in the **outArgs** property set
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionGetClipPreferences

This action allows a plugin to dynamically specify its preferences for input and output clips. Please see Image Effect Clip Preferences for more details on the behaviour. Clip preferences are constant for the duration of an effect, so this action need only be called once per clip, not once per frame.

This should be called once after creation of an instance, each time an input clip is changed, and whenever a parameter named in the *kOfxImageEffectPropClipPreferencesSlaveParam* has its value changed.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – has the following properties which the plugin can set
 - a set of char * X 1 properties, one for each of the input clips currently attached and the output clip, labelled with *OfxImageClipPropComponents_* post pended with the clip's name. This must be set to one of the component types which the host supports and the effect stated it can accept on that input
 - a set of char * X 1 properties, one for each of the input clips currently attached and the output clip, labelled with *OfxImageClipPropDepth_* post pended with the clip's name. This must be set to one of the pixel depths both the host and plugin supports
 - a set of double X 1 properties, one for each of the input clips currently attached and the output clip, labelled with *OfxImageClipPropPAR_* post pended with the clip's name. This is the pixel aspect ratio of the input and output clips. This must be set to a positive non zero double value,
 - *kOfxImageEffectPropFrameRate* the frame rate of the output clip, this must be set to a positive non zero double value
 - *kOfxImageClipPropFieldOrder* the fielding of the output clip
 - *kOfxImageEffectPropPreMultiplication* the premultiplication of the output clip

- *kOfxImageClipPropContinuousSamples* whether the output clip can produce different images at non-frame intervals, defaults to false,
- *kOfxImageEffectFrameVarying* whether the output clip can produce different images at different times, even if all parameters and inputs are constant, defaults to false.

Returns

- *kOfxStatOK*, the action was trapped and at least one of the properties in the outArgs was changed from its default value
- *kOfxStatReplyDefault*, the action was not trapped and the host should use the default values
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionIsIdentity

Sometimes an effect can pass through an input unprocessed, for example a blur effect with a blur size of 0. This action can be called by a host before it attempts to render an effect to determine if it can simply copy input directly to output without having to call the render action on the effect.

If the effect does not need to process any pixels, it should set the value of the *kOfxPropName* to the clip that the host should use as the output instead, and the *kOfxPropTime* property on outArgs to be the time at which the frame should be fetched from a clip.

The default action is to call the render action on the effect.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the time at which to test for identity
 - *kOfxImageEffectPropFieldToRender* the field to test for identity
 - *kOfxImageEffectPropRenderWindow* the window (in \ref PixelCoordinates) to test for identity under
 - *kOfxImageEffectPropRenderScale* the scale factor being applied to the images being rendered
- **outArgs** – has the following properties which the plugin can set
 - *kOfxPropName* this to the name of the clip that should be used if the effect is an identity transform, defaults to the empty string
 - *kOfxPropTime* the time to use from the indicated source clip as an identity image (allowing time slips to happen), defaults to the value in *kOfxPropTime* in inArgs

Returns

- *kOfxStatOK*, the action was trapped and the effect should not have its render action called, the values in outArgs indicate what frame from which clip to use instead
- *kOfxStatReplyDefault*, the action was not trapped and the host should call the render action
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionRender

This action is where an effect gets to push pixels and turn its input clips and parameter set into an output image. This is possibly quite complicated and covered in the Rendering Image Effects chapter.

The render action *must* be trapped by the plug-in, it cannot return *kOfxStatReplyDefault*. The pixels needs to be pushed I'm afraid.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxPropTime* the time at which to render
 - *kOfxImageEffectPropFieldToRender* the field to render
 - *kOfxImageEffectPropRenderWindow* the window (in \ref PixelCoordinates) to render
 - *kOfxImageEffectPropRenderScale* the scale factor being applied to the images being rendered
 - *kOfxImageEffectPropSequentialRenderStatus* whether the effect is currently being rendered in strict frame order on a single instance
 - *kOfxImageEffectPropInteractiveRenderStatus* if the render is in response to a user modifying the effect in an interactive session
 - *kOfxImageEffectPropRenderQualityDraft* if the render should be done in draft mode (e.g. for faster scrubbing)
- **outArgs** – is redundant and should be set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance
- *kOfxImageEffectActionBeginSequenceRender* has been called on the instance

Post

- *kOfxImageEffectActionEndSequenceRender* action will be called on the instance

Returns

- *kOfxStatOK*, the effect rendered happily
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectActionBeginSequenceRender

This action is passed to an image effect before it renders a range of frames. It is there to allow an effect to set things up for a long sequence of frames. Note that this is still called, even if only a single frame is being rendered in an interactive environment.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxImageEffectPropFrameRange* the range of frames (inclusive) that will be rendered

- *kOfxImageEffectPropFrameStep* what is the step between frames, generally set to 1 (for full frame renders) or 0.5 (for fielded renders)
- *kOfxPropIsInteractive* is this a single frame render due to user interaction in a GUI, or a proper full sequence render.
- *kOfxImageEffectPropRenderScale* the scale factor to apply to images for this call
- *kOfxImageEffectPropSequentialRenderStatus* whether the effect is currently being rendered in strict frame order on a single instance
- *kOfxImageEffectPropInteractiveRenderStatus* if the render is in response to a user modifying the effect in an interactive session
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance

Post

- *kOfxImageEffectActionRender* action will be called at least once on the instance
- *kOfxImageEffectActionEndSequenceRender* action will be called on the instance

Returns

- *kOfxStatOK*, the action was trapped and handled cleanly by the effect,
- *kOfxStatReplyDefault*, the action was not trapped, but all is well anyway,
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge,
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message,
- *kOfxStatErrFatal*

kOfxImageEffectActionEndSequenceRender

This action is passed to an image effect after it has rendered a range of frames. It is there to allow an effect to free resources after a long sequence of frame renders. Note that this is still called, even if only a single frame is being rendered in an interactive environment.

Parameters

- **handle** – handle to the instance, cast to an *OfxImageEffectHandle*
- **inArgs** – has the following properties
 - *kOfxImageEffectPropFrameRange* the range of frames (inclusive) that will be rendered
 - *kOfxImageEffectPropFrameStep* what is the step between frames, generally set to 1 (for full frame renders) or 0.5 (for fielded renders),
 - *kOfxPropIsInteractive*
 - is this a single frame render due to user interaction in a GUI, or a proper full sequence render.
 - *kOfxImageEffectPropRenderScale*
 - the scale factor to apply to images for this call
 - *kOfxImageEffectPropSequentialRenderStatus*
 - whether the effect is currently being rendered in strict frame order on a single instance

- *kOfxImageEffectPropInteractiveRenderStatus*
- if the render is in response to a user modifying the effect in an interactive session
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance
- *kOfxImageEffectActionEndSequenceRender* action was called on the instance
- *kOfxImageEffectActionRender* action was called at least once on the instance

Returns

- *kOfxStatOK*, the action was trapped and handled cleanly by the effect,
- *kOfxStatReplyDefault*, the action was not trapped, but all is well anyway,
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge,
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message,
- *kOfxStatErrFatal*

kOfxImageEffectActionDescribeInContext

This action is unique to OFX Image Effect plug-ins. Because a plugin is able to exhibit different behaviour depending on the context of use, each separate context will need to be described individually. It is within this action that image effects describe which parameters and input clips it requires.

This action will be called multiple times, one for each of the contexts the plugin says it is capable of implementing. If a host does not support a certain context, then it need not call *kOfxImageEffectActionDescribeInContext* for that context.

This action *must* be trapped, it is not optional.

Parameters

- **handle** – handle to the context descriptor, cast to an *OfxImageEffectHandle* this may or may not be the same as passed to *kOfxActionDescribe*
- **inArgs** – has the following property:
 - *kOfxImageEffectPropContext* the context being described
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionDescribe* has been called on the descriptor handle,
- *kOfxActionCreateInstance* has not been called

Returns

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatErrMissingHostFeature*, in which the context will be ignored by the host, the plugin may post a message
- *kOfxStatErrMemory*, in which case the action may be called again after a memory purge
- *kOfxStatFailed*, something wrong, but no error code appropriate, plugin to post message
- *kOfxStatErrFatal*

kOfxImageEffectPropSupportedContexts

Indicates to the host the contexts a plugin can be used in.

- Type - string X N
- Property Set - image effect descriptor passed to kOfxActionDescribe (read/write)
- Default - this has no defaults, it must be set
- Valid Values - This must be one of
 - *kOfxImageEffectContextGenerator*
 - *kOfxImageEffectContextFilter*
 - *kOfxImageEffectContextTransition*
 - *kOfxImageEffectContextPaint*
 - *kOfxImageEffectContextGeneral*
 - *kOfxImageEffectContextRetimer*

kOfxImageEffectPropPluginHandle

The plugin handle passed to the initial ‘describe’ action.

- Type - pointer X 1
- Property Set - plugin instance, (read only)

This value will be the same for all instances of a plugin.

kOfxImageEffectHostPropIsBackground

Indicates if a host is a background render.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 if the host is a foreground host, it may open the effect in an interactive session (or not)
 - 1 if the host is a background ‘processing only’ host, and the effect will never be opened in an interactive session.

kOfxImageEffectPluginPropSingleInstance

Indicates whether only one instance of a plugin can exist at the same time.

- Type - int X 1
- Property Set - plugin descriptor (read/write)
- Default - 0
- Valid Values - This must be one of

- 0 - which means multiple instances can exist simultaneously,
- 1 - which means only one instance can exist at any one time.

Some plugins, for whatever reason, may only be able to have a single instance in existence at any one time. This plugin property is used to indicate that.

kOfxImageEffectPluginRenderThreadSafety

Indicates how many simultaneous renders the plugin can deal with.

- Type - string X 1
- Property Set - plugin descriptor (read/write)
- Default - *kOfxImageEffectRenderInstanceSafe*
- Valid Values - This must be one of
 - *kOfxImageEffectRenderUnsafe* - indicating that only a single 'render' call can be made at any time among all instances,
 - *kOfxImageEffectRenderInstanceSafe* - indicating that any instance can have a single 'render' call at any one time,
 - *kOfxImageEffectRenderFullySafe* - indicating that any instance of a plugin can have multiple renders running simultaneously

kOfxImageEffectRenderUnsafe

String used to label render threads as un thread safe, see, *kOfxImageEffectPluginRenderThreadSafety*.

kOfxImageEffectRenderInstanceSafe

String used to label render threads as instance thread safe, *kOfxImageEffectPluginRenderThreadSafety*.

kOfxImageEffectRenderFullySafe

String used to label render threads as fully thread safe, *kOfxImageEffectPluginRenderThreadSafety*.

kOfxImageEffectPluginPropHostFrameThreading

Indicates whether a plugin lets the host perform per frame SMP threading.

- Type - int X 1
- Property Set - plugin descriptor (read/write)
- Default - 1
- Valid Values - This must be one of
 - 0 - which means that the plugin will perform any per frame SMP threading
 - 1 - which means the host can call an instance's render function simultaneously at the same frame, but with different windows to render.

kOfxImageEffectPropSupportsMultipleClipDepths

Indicates whether a host or plugin can support clips of differing component depths going into/out of an effect.

- Type - int X 1
- Property Set - plugin descriptor (read/write), host descriptor (read only)
- Default - 0 for a plugin
- Valid Values - This must be one of
 - 0 - in which case the host or plugin does not support clips of multiple pixel depths,
 - 1 - which means a host or plugin is able to to deal with clips of multiple pixel depths,

If a host indicates that it can support multiple pixels depths, then it will allow the plugin to explicitly set the output clip's pixel depth in the *kOfxImageEffectActionGetClipPreferences* action. See ImageEffectClipPreferences.

kOfxImageEffectPropSupportsMultipleClipPARs

Indicates whether a host or plugin can support clips of differing pixel aspect ratios going into/out of an effect.

- Type - int X 1
- Property Set - plugin descriptor (read/write), host descriptor (read only)
- Default - 0 for a plugin
- Valid Values - This must be one of
 - 0 - in which case the host or plugin does not support clips of multiple pixel aspect ratios
 - 1 - which means a host or plugin is able to to deal with clips of multiple pixel aspect ratios

If a host indicates that it can support multiple pixel aspect ratios, then it will allow the plugin to explicitly set the output clip's aspect ratio in the *kOfxImageEffectActionGetClipPreferences* action. See ImageEffectClipPreferences.

kOfxImageEffectPropClipPreferencesSlaveParam

Indicates the set of parameters on which a value change will trigger a change to clip preferences.

- Type - string X N
- Property Set - plugin descriptor (read/write)
- Default - none set
- Valid Values - the name of any described parameter

The plugin uses this to inform the host of the subset of parameters that affect the effect's clip preferences. A value change in any one of these will trigger a call to the clip preferences action.

The plugin can be slaved to multiple parameters (setting index 0, then index 1 etc...)

kOfxImageEffectPropSetableFrameRate

Indicates whether the host will let a plugin set the frame rate of the output clip.

- Type - int X 1

- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 - in which case the plugin may not change the frame rate of the output clip,
 - 1 - which means a plugin is able to change the output clip's frame rate in the *kOfxImageEffectActionGetClipPreferences* action.

See ImageEffectClipPreferences.

If a clip can be continuously sampled, the frame rate will be set to 0.

kOfxImageEffectPropSettableFielding

Indicates whether the host will let a plugin set the fielding of the output clip.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 - in which case the plugin may not change the fielding of the output clip,
 - 1 - which means a plugin is able to change the output clip's fielding in the *kOfxImageEffectActionGetClipPreferences* action.

See ImageEffectClipPreferences.

kOfxImageEffectInstancePropSequentialRender

Indicates whether a plugin needs sequential rendering, and a host support it.

- Type - int X 1
- Property Set - plugin descriptor (read/write) or plugin instance (read/write), and host descriptor (read only)
- Default - 0
- Valid Values -
 - 0 - for a plugin, indicates that a plugin does not need to be sequentially rendered to be correct, for a host, indicates that it cannot ever guarantee sequential rendering,
 - 1 - for a plugin, indicates that it needs to be sequentially rendered to be correct, for a host, indicates that it can always support sequential rendering of plugins that are sequentially rendered,
 - 2 - for a plugin, indicates that it is best to render sequentially, but will still produce correct results if not, for a host, indicates that it can sometimes render sequentially, and will have set *kOfxImageEffectPropSequentialRenderStatus* on the relevant actions

Some effects have temporal dependancies, some information from from the rendering of frame N-1 is needed to render frame N correctly. This property is set by an effect to indicate such a situation. Also, some effects are more efficient if they run sequentially, but can still render correct images even if they do not, eg: a complex particle system.

During an interactive session a host may attempt to render a frame out of sequence (for example when the user scrubs the current time), and the effect needs to deal with such a situation as best it can to provide feedback to the user.

However if a host caches output, any frame frame generated in random temporal order needs to be considered invalid and needs to be re-rendered when the host finally performs a first to last render of the output sequence.

In all cases, a host will set the `kOfxImageEffectPropSequentialRenderStatus` flag to indicate its sequential render status.

kOfxImageEffectPropSequentialRenderStatus

Property on all the render action that indicate the current sequential render status of a host.

- Type - int X 1
- Property Set - read only property on the inArgs of the following actions...
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values -
 - 0 - the host is not currently sequentially rendering,
 - 1 - the host is currently rendering in a way so that it guarantees sequential rendering.

This property is set to indicate whether the effect is currently being rendered in frame order on a single effect instance. See *kOfxImageEffectInstancePropSequentialRender* for more details on sequential rendering.

kOfxHostNativeOriginBottomLeft

kOfxHostNativeOriginTopLeft

kOfxHostNativeOriginCenter

kOfxImageEffectHostPropNativeOrigin

Property that indicates the host native UI space - this is only a UI hint, has no impact on pixel processing.

- Type - UTF8 string X 1
- Property Set - read only property (host)
 - Valid Values - “kOfxImageEffectHostPropNativeOriginBottomLeft” - 0,0 bottom left “kOfxImageEffectHostPropNativeOriginTopLeft” - 0,0 top left “kOfxImageEffectHostPropNativeOriginCenter” - 0,0 center (screen space)

This property is set to `kOfxHostNativeOriginBottomLeft` pre V1.4 and was to be discovered by plug-ins. This is useful for drawing overlay for points... so everything matches the rest of the app (for example expression linking to other tools, or simply match the reported location of the host viewer).

kOfxImageEffectPropInteractiveRenderStatus

Property that indicates if a plugin is being rendered in response to user interaction.

- Type - int X 1

- Property Set - read only property on the inArgs of the following actions...
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values -
 - 0 - the host is rendering the instance due to some reason other than an interactive tweak on a UI,
 - 1 - the instance is being rendered because a user is modifying parameters in an interactive session.

This property is set to 1 on all render calls that have been triggered because a user is actively modifying an effect (or up stream effect) in an interactive session. This typically means that the effect is not being rendered as a part of a sequence, but as a single frame.

kOfxImageEffectPluginPropGrouping

Indicates the effect group for this plugin.

- Type - UTF8 string X 1
- Property Set - plugin descriptor (read/write)
- Default - ""

This is purely a user interface hint for the host so it can group related effects on any menus it may have.

kOfxImageEffectPropSupportsOverlays

Indicates whether a host support image effect ImageEffectOverlays.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - This must be one of
 - 0 - the host won't allow a plugin to draw a GUI over the output image,
 - 1 - the host will allow a plugin to draw a GUI over the output image.

kOfxImageEffectPluginPropOverlayInteractV1

Sets the entry for an effect's overlay interaction.

- Type - pointer X 1
- Property Set - plugin descriptor (read/write)
- Default - NULL
- Valid Values - must point to an *OfxPluginEntryPoint*

The entry point pointed to must be one that handles custom interaction actions.

kOfxImageEffectPluginPropOverlayInteractV2

Sets the entry for an effect's overlay interaction. Unlike `kOfxImageEffectPluginPropOverlayInteractV1`, the overlay interact in the plug-in is expected to implement the `kOfxInteractActionDraw` using the *OfxDrawSuiteV1*.

- Type - pointer X 1
- Property Set - plugin descriptor (read/write)
- Default - NULL
- Valid Values - must point to an *OfxPluginEntryPoint*

The entry point pointed to must be one that handles custom interaction actions.

kOfxImageEffectPropSupportsMultiResolution

Indicates whether a plugin or host support multiple resolution images.

- Type - int X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - 1 for plugins
- Valid Values - This must be one of
 - 0 - the plugin or host does not support multiple resolutions
 - 1 - the plugin or host does support multiple resolutions

Multiple resolution images mean...

- input and output images can be of any size
- input and output images can be offset from the origin

kOfxImageEffectPropSupportsTiles

Indicates whether a clip, plugin or host supports tiled images.

- Type - int X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write), clip descriptor (read/write), instance (read/write)
- Default - to 1 for a plugin and clip
- Valid Values - This must be one of 0 or 1

Tiled images mean that input or output images can contain pixel data that is only a subset of their full RoD.

If a clip or plugin does not support tiled images, then the host should supply full RoD images to the effect whenever it fetches one.

V1.4: It is now possible (defined) to change `OfxImageEffectPropSupportsTiles` in Instance Changed

kOfxImageEffectPropTemporalClipAccess

Indicates support for random temporal access to images in a clip.

- Type - int X 1
- Property Set - host descriptor (read only), plugin descriptor (read/write), clip descriptor (read/write)
- Default - to 0 for a plugin and clip
- Valid Values - This must be one of 0 or 1

On a host, it indicates whether the host supports temporal access to images.

On a plugin, indicates if the plugin needs temporal access to images.

On a clip, it indicates that the clip needs temporal access to images.

kOfxImageEffectPropContext

Indicates the context a plugin instance has been created for.

- Type - string X 1
- Property Set - image effect instance (read only)
- Valid Values - This must be one of
 - *kOfxImageEffectContextGenerator*
 - *kOfxImageEffectContextFilter*
 - *kOfxImageEffectContextTransition*
 - *kOfxImageEffectContextPaint*
 - *kOfxImageEffectContextGeneral*
 - *kOfxImageEffectContextRetimer*

kOfxImageEffectPropPixelDepth

Indicates the type of each component in a clip or image (after any mapping)

- Type - string X 1
- Property Set - clip instance (read only), image instance (read only)
- Valid Values - This must be one of
 - *kOfxBitDepthNone* (implying a clip is unconnected, not valid for an image)
 - *kOfxBitDepthByte*
 - *kOfxBitDepthShort*
 - *kOfxBitDepthHalf*
 - *kOfxBitDepthFloat*

Note that for a clip, this is the value set by the clip preferences action, not the raw ‘actual’ value of the clip.

kOfxImageEffectPropComponents

Indicates the current component type in a clip or image (after any mapping)

- Type - string X 1
- Property Set - clip instance (read only), image instance (read only)
- Valid Values - This must be one of
 - kOfxImageComponentNone (implying a clip is unconnected, not valid for an image)
 - kOfxImageComponentRGBA
 - kOfxImageComponentRGB
 - kOfxImageComponentAlpha

Note that for a clip, this is the value set by the clip preferences action, not the raw ‘actual’ value of the clip.

kOfxImagePropUniqueIdentifier

Uniquely labels an image.

- Type - ASCII string X 1
- Property Set - image instance (read only)

This is host set and allows a plug-in to differentiate between images. This is especially useful if a plugin caches analysed information about the image (for example motion vectors). The plugin can label the cached information with this identifier. If a user connects a different clip to the analysed input, or the image has changed in some way then the plugin can detect this via an identifier change and re-evaluate the cached information.

kOfxImageClipPropContinuousSamples

Clip and action argument property which indicates that the clip can be sampled continuously.

- Type - int X 1
- Property Set - clip instance (read only), as an out argument to *kOfxImageEffectActionGetClipPreferences* action (read/write)
- Default - 0 as an out argument to the *kOfxImageEffectActionGetClipPreferences* action
- Valid Values - This must be one of...
 - 0 if the images can only be sampled at discreet times (eg: the clip is a sequence of frames),
 - 1 if the images can only be sampled continuously (eg: the clip is infact an animating roto spline and can be rendered anywhen).

If this is set to true, then the frame rate of a clip is effectively infinite, so to stop arithmetic errors the frame rate should then be set to 0.

kOfxImageClipPropUnmappedPixelDepth

Indicates the type of each component in a clip before any mapping by clip preferences.

- Type - string X 1
- Property Set - clip instance (read only)
- Valid Values - This must be one of
 - kOfxBitDepthNone (implying a clip is unconnected image)
 - kOfxBitDepthByte
 - kOfxBitDepthShort
 - kOfxBitDepthHalf
 - kOfxBitDepthFloat

This is the actual value of the component depth, before any mapping by clip preferences.

kOfxImageClipPropUnmappedComponents

Indicates the current 'raw' component type on a clip before any mapping by clip preferences.

- Type - string X 1
- Property Set - clip instance (read only),
- Valid Values - This must be one of
 - kOfxImageComponentNone (implying a clip is unconnected)
 - kOfxImageComponentRGBA
 - kOfxImageComponentRGB
 - kOfxImageComponentAlpha

kOfxImageEffectPropPreMultiplication

Indicates the premultiplication state of a clip or image.

- Type - string X 1
- Property Set - clip instance (read only), image instance (read only), out args property in the *kOfxImageEffectActionGetClipPreferences* action (read/write)
- Valid Values - This must be one of
 - kOfxImageOpaque - the image is opaque and so has no premultiplication state
 - kOfxImagePreMultiplied - the image is premultiplied by its alpha
 - kOfxImageUnPreMultiplied - the image is unpremultiplied

See the documentation on clip preferences for more details on how this is used with the *kOfxImageEffectActionGetClipPreferences* action.

kOfxImageOpaque

Used to flag the alpha of an image as opaque

kOfxImagePreMultiplied

Used to flag an image as premultiplied

kOfxImageUnPreMultiplied

Used to flag an image as unpremultiplied

kOfxImageEffectPropSupportedPixelDepths

Indicates the bit depths support by a plug-in or host.

- Type - string X N
- Property Set - host descriptor (read only), plugin descriptor (read/write)
- Default - plugin descriptor none set
- Valid Values - This must be one of
 - kOfxBitDepthNone (implying a clip is unconnected, not valid for an image)
 - kOfxBitDepthByte
 - kOfxBitDepthShort
 - kOfxBitDepthHalf
 - kOfxBitDepthFloat

The default for a plugin is to have none set, the plugin *must* define at least one in its describe action.

kOfxImageEffectPropSupportedComponents

Indicates the components supported by a clip or host,.

- Type - string X N
- Property Set - host descriptor (read only), clip descriptor (read/write)
- Valid Values - This must be one of
 - kOfxImageComponentNone (implying a clip is unconnected)
 - kOfxImageComponentRGBA
 - kOfxImageComponentRGB
 - kOfxImageComponentAlpha

This list of strings indicate what component types are supported by a host or are expected as input to a clip.

The default for a clip descriptor is to have none set, the plugin *must* define at least one in its define function

kOfxImageClipPropOptional

Indicates if a clip is optional.

- Type - int X 1
- Property Set - clip descriptor (read/write)
- Default - 0
- Valid Values - This must be one of 0 or 1

kOfxImageClipPropIsMask

Indicates that a clip is intended to be used as a mask input.

- Type - int X 1
- Property Set - clip descriptor (read/write)
- Default - 0
- Valid Values - This must be one of 0 or 1

Set this property on any clip which will only ever have single channel alpha images fetched from it. Typically on an optional clip such as a junk matte in a keyer.

This property acts as a hint to hosts indicating that they could feed the effect from a roto shape (or similar) rather than an 'ordinary' clip.

kOfxImagePropPixelAspectRatio

The pixel aspect ratio of a clip or image.

- Type - double X 1
- Property Set - clip instance (read only), image instance (read only) and *kOfxImageEffectActionGetClipPreferences* action out args property (read/write)

kOfxImageEffectPropFrameRate

The frame rate of a clip or instance's project.

- Type - double X 1
- Property Set - clip instance (read only), effect instance (read only) and *kOfxImageEffectActionGetClipPreferences* action out args property (read/write)

For an input clip this is the frame rate of the clip.

For an output clip, the frame rate mapped via pixel preferences.

For an instance, this is the frame rate of the project the effect is in.

For the outargs property in the *kOfxImageEffectActionGetClipPreferences* action, it is used to change the frame rate of the output clip.

kOfxImageEffectPropUnmappedFrameRate

Indicates the original unmapped frame rate (frames/second) of a clip.

- Type - double X 1
- Property Set - clip instance (read only),

If a plugin changes the output frame rate in the pixel preferences action, this property allows a plugin to get to the original value.

kOfxImageEffectPropFrameStep

The frame step used for a sequence of renders.

- Type - double X 1
- Property Set - an in argument for the *kOfxImageEffectActionBeginSequenceRender* action (read only)
- Valid Values - can be any positive value, but typically
 - 1 for frame based material
 - 0.5 for field based material

kOfxImageEffectPropFrameRange

The frame range over which a clip has images.

- Type - double X 2
- Property Set - clip instance (read only)

Dimension 0 is the first frame for which the clip can produce valid data.

Dimension 1 is the last frame for which the clip can produce valid data.

kOfxImageEffectPropUnmappedFrameRange

The unmaped frame range over which an output clip has images.

- Type - double X 2
- Property Set - clip instance (read only)

Dimension 0 is the first frame for which the clip can produce valid data.

Dimension 1 is the last frame for which the clip can produce valid data.

If a plugin changes the output frame rate in the pixel preferences action, it will affect the frame range of the output clip, this property allows a plugin to get to the original value.

kOfxImageClipPropConnected

Says whether the clip is actually connected at the moment.

- Type - int X 1
- Property Set - clip instance (read only)
- Valid Values - This must be one of 0 or 1

An instance may have a clip may not be connected to an object that can produce image data. Use this to find out.

Any clip that is not optional will *always* be connected during a render action. However, during interface actions, even non optional clips may be unconnected.

kOfxImageEffectFrameVarying

Indicates whether an effect will generate different images from frame to frame.

- Type - int X 1
- Property Set - out argument to *kOfxImageEffectActionGetClipPreferences* action (read/write).
- Default - 0
- Valid Values - This must be one of 0 or 1

This property indicates whether a plugin will generate a different image from frame to frame, even if no parameters or input image changes. For example a generator that creates random noise pixel at each frame.

kOfxImageEffectPropRenderScale

The proxy render scale currently being applied.

- Type - double X 2
- Property Set - an image instance (read only) and as read only an in argument on the following actions,
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
 - *kOfxImageEffectActionIsIdentity*
 - *kOfxImageEffectActionGetRegionOfDefinition*
 - *kOfxImageEffectActionGetRegionsOfInterest*
 - *kOfxActionInstanceChanged*
 - *kOfxInteractActionDraw*
 - *kOfxInteractActionPenMotion*
 - *kOfxInteractActionPenDown*
 - *kOfxInteractActionPenUp*
 - *kOfxInteractActionKeyDown*
 - *kOfxInteractActionKeyUp*
 - *kOfxInteractActionKeyRepeat*
 - *kOfxInteractActionGainFocus*
 - *kOfxInteractActionLoseFocus*

This should be applied to any spatial parameters to position them correctly. Not that the 'x' value does not include any pixel aspect ratios.

kOfxImageEffectPropRenderQualityDraft

Indicates whether an effect can take quality shortcuts to improve speed.

- Type - int X 1

- Property Set - render calls, host (read-only)
- Default - 0 - 0: Best Quality (1: Draft)
- Valid Values - This must be one of 0 or 1

This property indicates that the host provides the plug-in the option to render in Draft/Preview mode. This is useful for applications that must support fast scrubbing. These allow a plug-in to take short-cuts for improved performance when the situation allows and it makes sense, for example to generate thumbnails with effects applied. For example switch to a cheaper interpolation type or rendering mode. A plugin should expect frames rendered in this manner that will not be stuck in host cache unless the cache is only used in the same draft situations. If an host does not support that property a value of 0 is assumed. Also note that some hosts do implement `kOfxImageEffectPropRenderScale` - these two properties can be used independently.

kOfxImageEffectPropProjectExtent

The extent of the current project in canonical coordinates.

- Type - double X 2
- Property Set - a plugin instance (read only)

The extent is the size of the 'output' for the current project. See `NormalisedCoordinateSystem` for more information on the project extent.

The extent is in canonical coordinates and only returns the top right position, as the extent is always rooted at 0,0.

For example a PAL SD project would have an extent of 768, 576.

kOfxImageEffectPropProjectSize

The size of the current project in canonical coordinates.

- Type - double X 2
- Property Set - a plugin instance (read only)

The size of a project is a sub set of the `kOfxImageEffectPropProjectExtent`. For example a project may be a PAL SD project, but only be a letter-box within that. The project size is the size of this sub window.

The project size is in canonical coordinates.

See `NormalisedCoordinateSystem` for more information on the project extent.

kOfxImageEffectPropProjectOffset

The offset of the current project in canonical coordinates.

- Type - double X 2
- Property Set - a plugin instance (read only)

The offset is related to the `kOfxImageEffectPropProjectSize` and is the offset from the origin of the project 'subwindow'.

For example for a PAL SD project that is in letterbox form, the project offset is the offset to the bottom left hand corner of the letter box.

The project offset is in canonical coordinates.

See `NormalisedCoordinateSystem` for more information on the project extent.

kOfxImageEffectPropProjectPixelAspectRatio

The pixel aspect ratio of the current project.

- Type - double X 1
- Property Set - a plugin instance (read only)

kOfxImageEffectInstancePropEffectDuration

The duration of the effect.

- Type - double X 1
- Property Set - a plugin instance (read only)

This contains the duration of the plug-in effect, in frames.

kOfxImageClipPropFieldOrder

Which spatial field occurs temporally first in a frame.

- Type - string X 1
- Property Set - a clip instance (read only)
- Valid Values - This must be one of
 - *kOfxImageFieldNone* - the material is unfielded
 - *kOfxImageFieldLower* - the material is fielded, with image rows 0,2,4... occurring first in a frame
 - *kOfxImageFieldUpper* - the material is fielded, with image rows line 1,3,5... occurring first in a frame

kOfxImagePropData

The pixel data pointer of an image.

- Type - pointer X 1
- Property Set - an image instance (read only)

This property contains one of:

- a pointer to memory that is the lower left hand corner of an image
- a pointer to CUDA memory, if the Render action arguments includes `kOfxImageEffectPropCudaEnabled=1`
- an `id<MTLBuffer>`, if the Render action arguments includes `kOfxImageEffectPropMetalEnabled=1`
- a `cl_mem`, if the Render action arguments includes `kOfxImageEffectPropOpenCLEnabled=1`

See *kOfxImageEffectPropCudaEnabled*, *kOfxImageEffectPropMetalEnabled* and *kOfxImageEffectPropOpenCLEnabled*

kOfxImagePropBounds

The bounds of an image's pixels.

- Type - integer X 4
- Property Set - an image instance (read only)

The bounds, in PixelCoordinates, are of the addressable pixels in an image's data pointer.

The order of the values is x1, y1, x2, y2.

X values are $x1 \leq X < x2$ Y values are $y1 \leq Y < y2$

For less than full frame images, the pixel bounds will be contained by the *kOfxImagePropRegionOfDefinition* bounds.

kOfxImagePropRegionOfDefinition

The full region of definition of an image.

- Type - integer X 4
- Property Set - an image instance (read only)

An image's region of definition, in PixelCoordinates, is the full frame area of the image plane that the image covers.

The order of the values is x1, y1, x2, y2.

X values are $x1 \leq X < x2$ Y values are $y1 \leq Y < y2$

The *kOfxImagePropBounds* property contains the actual addressable pixels in an image, which may be less than its full region of definition.

kOfxImagePropRowBytes

The number of bytes in a row of an image.

- Type - integer X 1
- Property Set - an image instance (read only)

For various alignment reasons, a row of pixels may need to be padded at the end with several bytes before the next row starts in memory.

This property indicates the number of bytes in a row of pixels. This will be at least $\text{sizeof}(\text{PIXEL}) * (\text{bounds.x2} - \text{bounds.x1})$. Where bounds is fetched from the *kOfxImagePropBounds* property.

Note that (for CPU images only, not CUDA/Metal/OpenCL Buffers, nor OpenGL textures accessed via the OpenGL Render Suite) row bytes can be negative, which allows hosts with a native top down row order to pass image into OFX without having to repack pixels. Row bytes is not supported for OpenCL Images.

kOfxImagePropField

Which fields are present in the image.

- Type - string X 1

- Property Set - an image instance (read only)
- Valid Values - This must be one of
 - *kOfxImageFieldNone* - the image is an unfielded frame
 - *kOfxImageFieldBoth* - the image is fielded and contains both interlaced fields
 - *kOfxImageFieldLower* - the image is fielded and contains a single field, being the lower field (rows 0,2,4...)
 - *kOfxImageFieldUpper* - the image is fielded and contains a single field, being the upper field (rows 1,3,5...)

kOfxImageEffectPluginPropFieldRenderTwiceAlways

Controls how a plugin renders fielded footage.

- Type - integer X 1
- Property Set - a plugin descriptor (read/write)
- Default - 1
- Valid Values - This must be one of
 - 0 - the plugin is to have its render function called twice, only if there is animation in any of its parameters
 - 1 - the plugin is to have its render function called twice always

kOfxImageClipPropFieldExtraction

Controls how a plugin fetched fielded imagery from a clip.

- Type - string X 1
- Property Set - a clip descriptor (read/write)
- Default - *kOfxImageFieldDoubled*
- Valid Values - This must be one of
 - *kOfxImageFieldBoth* - fetch a full frame interlaced image
 - *kOfxImageFieldSingle* - fetch a single field, making a half height image
 - *kOfxImageFieldDoubled* - fetch a single field, but doubling each line and so making a full height image

This controls how a plug-in wishes to fetch images from a fielded clip, so it can tune its behaviour when it renders fielded footage.

Note that if it fetches *kOfxImageFieldSingle* and the host stores images natively as both fields interlaced, it can return a single image by doubling rowbytes and tweaking the starting address of the image data. This saves on a buffer copy.

kOfxImageEffectPropFieldToRender

Indicates which field is being rendered.

- Type - string X 1
- Property Set - a read only in argument property to *kOfxImageEffectActionRender* and *kOfxImageEffectActionIsIdentity*
- Valid Values - this must be one of
 - kOfxImageFieldNone - there are no fields to deal with, all images are full frame
 - kOfxImageFieldBoth - the imagery is fielded and both scan lines should be rendered
 - kOfxImageFieldLower - the lower field is being rendered (lines 0,2,4...)
 - kOfxImageFieldUpper - the upper field is being rendered (lines 1,3,5...)

kOfxImageEffectPropRegionOfDefinition

Used to indicate the region of definition of a plug-in.

- Type - double X 4
- Property Set - a read/write out argument property to the *kOfxImageEffectActionGetRegionOfDefinition* action
- Default - see *kOfxImageEffectActionGetRegionOfDefinition*

The order of the values is x1, y1, x2, y2.

This will be in CanonicalCoordinates

kOfxImageEffectPropRegionOfInterest

The value of a region of interest.

- Type - double X 4
- Property Set - a read only in argument property to the *kOfxImageEffectActionGetRegionsOfInterest* action

A host passes this value into the region of interest action to specify the region it is interested in rendering.

The order of the values is x1, y1, x2, y2.

This will be in CanonicalCoordinates.

kOfxImageEffectPropRenderWindow

The region to be rendered.

- Type - integer X 4
- Property Set - a read only in argument property to the *kOfxImageEffectActionRender* and *kOfxImageEffectActionIsIdentity* actions

The order of the values is x1, y1, x2, y2.

This will be in PixelCoordinates

kOfxImageFieldNone

String used to label imagery as having no fields

kOfxImageFieldLower

String used to label the lower field (scan lines 0,2,4...) of fielded imagery

kOfxImageFieldUpper

String used to label the upper field (scan lines 1,3,5...) of fielded imagery

kOfxImageFieldBoth

String used to label both fields of fielded imagery, indicating interlaced footage

kOfxImageFieldSingle

String used to label an image that consists of a single field, and so is half height

kOfxImageFieldDoubled

String used to label an image that consists of a single field, but each scan line is double, and so is full height

kOfxImageEffectOutputClipName

String that is the name of the standard OFX output clip.

kOfxImageEffectSimpleSourceClipName

String that is the name of the standard OFX single source input clip.

kOfxImageEffectTransitionSourceFromClipName

String that is the name of the 'from' clip in the OFX transition context.

kOfxImageEffectTransitionSourceToClipName

String that is the name of the 'to' clip in the OFX transition context.

kOfxImageEffectTransitionParamName

the name of the mandated 'Transition' param for the transition context

kOfxImageEffectRetimerParamName

the name of the mandated 'SourceTime' param for the retimer context

kOfxImageEffectSuite

the string that names image effect suites, passed to *OfxHost::fetchSuite*

kOfxStatErrImageFormat

Error code for incorrect image formats.

Typedefs

typedef struct OfxImageEffectStruct ***OfxImageEffectHandle**

Blind declaration of an OFX image effect.

typedef struct OfxImageClipStruct ***OfxImageClipHandle**

Blind declaration of an OFX image effect.

typedef struct OfxImageMemoryStruct ***OfxImageMemoryHandle**

Blind declaration for an handle to image memory returned by the image memory management routines.

typedef struct *OfxImageEffectSuiteV1* **OfxImageEffectSuiteV1**

The OFX suite for image effects.

This suite provides the functions needed by a plugin to defined and use an image effect plugin.

file **ofxInteract.h**

#include "ofxCore.h" Contains the API for ofx plugin defined GUIs and interaction.

Defines

kOfxInteractSuite

kOfxInteractPropSlaveToParam

The set of parameters on which a value change will trigger a redraw for an interact.

- Type - string X N
- Property Set - interact instance property (read/write)
- Default - no values set
- Valid Values - the name of any parameter associated with this interact.

If the interact is representing the state of some set of OFX parameters, then it will need to be redrawn if any of those parameters' values change. This multi-dimensional property links such parameters to the interact.

The interact can be slaved to multiple parameters (setting index 0, then index 1 etc...)

kOfxInteractPropPixelScale

The size of a real screen pixel under the interact's canonical projection.

- Type - double X 2
- Property Set - interact instance and actions (read only)

kOfxInteractPropBackgroundColour

The background colour of the application behind an interact instance.

- Type - double X 3
- Property Set - read only on the interact instance and in argument to the *kOfxInteractActionDraw* action
- Valid Values - from 0 to 1

The components are in the order red, green then blue.

kOfxInteractPropSuggestedColour

The suggested colour to draw a widget in an interact, typically for overlays.

- Type - double X 3
- Property Set - read only on the interact instance
- Default - 1.0
- Valid Values - greater than or equal to 0.0

Some applications allow the user to specify colours of any overlay via a colour picker, this property represents the value of that colour. Plugins are at liberty to use this or not when they draw an overlay.

If a host does not support such a colour, it should return `kOfxStatReplyDefault`

kOfxInteractPropPenPosition

The position of the pen in an interact.

- Type - double X 2
- Property Set - read only in argument to the *kOfxInteractActionPenMotion*, *kOfxInteractActionPenDown* and *kOfxInteractActionPenUp* actions

This value passes the position of the pen into an interact. This is in the interact's canonical coordinates.

kOfxInteractPropPenViewportPosition

The position of the pen in an interact in viewport coordinates.

- Type - int X 2
- Property Set - read only in argument to the *kOfxInteractActionPenMotion*, *kOfxInteractActionPenDown* and *kOfxInteractActionPenUp* actions

This value passes the position of the pen into an interact. This is in the interact's OpenGL viewport coordinates, with 0,0 being at the bottom left.

kOfxInteractPropPenPressure

The pressure of the pen in an interact.

- Type - double X 1
- Property Set - read only in argument to the *kOfxInteractActionPenMotion*, *kOfxInteractActionPenDown* and *kOfxInteractActionPenUp* actions
- Valid Values - from 0 (no pressure) to 1 (maximum pressure)

This is used to indicate the status of the ‘pen’ in an interact. If a pen has only two states (eg: a mouse button), these should map to 0.0 and 1.0.

kOfxInteractPropBitDepth

Indicates whether the dits per component in the interact’s openGL frame buffer.

- Type - int X 1
- Property Set - interact instance and descriptor (read only)

kOfxInteractPropHasAlpha

Indicates whether the interact’s frame buffer has an alpha component or not.

- Type - int X 1
- Property Set - interact instance and descriptor (read only)
- Valid Values - This must be one of
 - 0 indicates no alpha component
 - 1 indicates an alpha component

kOfxActionDescribeInteract

This action is the first action passed to an interact. It is where an interact defines how it behaves and the resources it needs to function. If not trapped, the default action is for the host to carry on as normal Note that the handle passed in acts as a descriptor for, rather than an instance of the interact.

Parameters

- **handle** – handle to the interact descriptor, cast to an *OfxInteractHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- The plugin has been loaded and the effect described.

Returns

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatErrMemory* in which case describe may be called again after a memory purge
- *kOfxStatFailed* something was wrong, the host should ignore the interact
- *kOfxStatErrFatal*

kOfxActionCreateInstanceInteract

This action is the first action passed to an interact instance after its creation. It is there to allow a plugin to create any per-instance data structures it may need.

Parameters

- **handle** – handle to the interact instance, cast to an *OfxInteractHandle*
- **inArgs** – is redundant and is set to NULL

- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionDescribe* has been called on this interact

Post

- the instance pointer will be valid until the *kOfxActionDestroyInstance* action is passed to the plug-in with the same instance handle

Returns

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatReplyDefault* the action was ignored, but all was well anyway
- *kOfxStatErrFatal*
- *kOfxStatErrMemory* in which case this may be called again after a memory purge
- *kOfxStatFailed* in which case the host should ignore this interact

kOfxActionDestroyInstanceInteract

This action is the last passed to an interact's instance before its destruction. It is there to allow a plugin to destroy any per-instance data structures it may have created.

Parameters

- **handle** – handle to the interact instance, cast to an *OfxInteractHandle*
- **inArgs** – is redundant and is set to NULL
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the handle,
- the instance has not had any of its members destroyed yet

Post

- the instance pointer is no longer valid and any operation on it will be undefined

Returns

To some extent, what is returned is moot, a bit like throwing an exception in a C++ destructor, so the host should continue destruction of the instance regardless

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatReplyDefault* the action was ignored as the effect had nothing to do
- *kOfxStatErrFatal*
- *kOfxStatFailed* something went wrong, but no error code appropriate.

kOfxInteractActionDraw

This action is issued to an interact whenever the host needs the plugin to redraw the given interact.

The interact should either issue OpenGL calls to draw itself, or use DrawSuite calls.

If this is called via *kOfxImageEffectPluginPropOverlayInteractV2*, drawing MUST use DrawSuite.

If this is called via `kOfxImageEffectPluginPropOverlayInteractV1`, drawing SHOULD use OpenGL. Some existing plugins may use DrawSuite via `kOfxImageEffectPluginPropOverlayInteractV1` if it's supported by the host, but this is discouraged.

Note that the interact may (in the case of custom parameter GUIs) or may not (in the case of image effect overlays) be required to swap buffers, that is up to the kind of interact.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle
- the openGL context for this interact has been set
- the projection matrix will correspond to the interact's canonical view

Returns

- *kOfxStatOK* the action was trapped and all was well
- *kOfxStatReplyDefault* the action was ignored
- *kOfxStatErrFatal*
- *kOfxStatFailed* something went wrong, the host should ignore this interact in future

kOfxInteractActionPenMotion

This action is issued whenever the pen moves and the interact's has focus. It should be issued whether the pen is currently up or down. No openGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
 - *kOfxInteractPropPenPosition* position of the pen in,
 - *kOfxInteractPropPenViewPortPosition* position of the pen in,

- *kOfxInteractPropPenPressure* the pressure of the pen,
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK* the host should not pass the pen motion to any other interactive object it may own that shares the same view.

Returns

- *kOfxStatOK* the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault* the action was not trapped and the host can deal with it if it wants

kOfxInteractActionPenDown

This action is issued when a pen transitions for the ‘up’ to the ‘down’ state. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin,
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
 - *kOfxInteractPropPenPosition* position of the pen in
 - *kOfxInteractPropPenViewportPosition* position of the pen in
 - *kOfxInteractPropPenPressure* the pressure of the pen
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same view.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault*, the action was not trapped and the host can deal with it if it wants

kOfxInteractActionPenUp

This action is issued when a pen transitions for the ‘down’ to the ‘up’ state. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin,
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
 - *kOfxInteractPropPenPosition* position of the pen in
 - *kOfxInteractPropPenViewportPosition* position of the pen in
 - *kOfxInteractPropPenPressure* the pressure of the pen
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same view.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault*, the action was not trapped and the host can deal with it if it wants

kOfxInteractActionKeyDown

This action is issued when a key on the keyboard is depressed. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxPropKeySym* single integer value representing the key that was manipulated, this may not have a UTF8 representation (eg: a return key)
 - *kOfxPropKeyString* UTF8 string representing a character key that was pressed, some keys have no UTF8 encoding, in which case this is ""
 - *kOfxPropTime* the effect time at which changed occurred

- *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same focus.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault*, the action was not trapped and the host can deal with it if it wants

kOfxInteractActionKeyUp

This action is issued when a key on the keyboard is released. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxPropKeySym* single integer value representing the key that was manipulated, this may not have a UTF8 representation (eg: a return key)
 - *kOfxPropKeyString* UTF8 string representing a character key that was pressed, some keys have no UTF8 encoding, in which case this is ""
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same focus.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault*, the action was not trapped and the host can deal with it if it wants

kOfxInteractActionKeyRepeat

This action is issued when a key on the keyboard is repeated. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact has been,
 - *kOfxPropKeySym* single integer value representing the key that was manipulated, this may not have a UTF8 representation (eg: a return key)
 - *kOfxPropKeyString* UTF8 string representing a character key that was pressed, some keys have no UTF8 encoding, in which case this is ""
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,
- the current instance handle has had *kOfxInteractActionGainFocus* called on it

Post

- if the instance returns *kOfxStatOK*, the host should not pass the pen motion to any other interactive object it may own that shares the same focus.

Returns

- *kOfxStatOK*, the action was trapped and the host should not pass the event to other objects it may own
- *kOfxStatReplyDefault*, the action was not trapped and the host can deal with it if it wants

kOfxInteractActionGainFocus

This action is issued when an interact gains input focus. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact is being used on,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels,
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Returns

- *kOfxStatOK* the action was trapped
- *kOfxStatReplyDefault* the action was not trapped

kOfxInteractActionLoseFocus

This action is issued when an interact loses input focus. No OpenGL calls should be issued by the plug-in during this action.

Parameters

- **handle** – handle to an interact instance, cast to an *OfxInteractHandle*
- **inArgs** – has the following properties on an image effect plugin
 - *kOfxPropEffectInstance* a handle to the effect for which the interact is being used on,
 - *kOfxInteractPropPixelScale* the scale factor to convert canonical pixels to screen pixels,
 - *kOfxInteractPropBackgroundColour* the background colour of the application behind the current view
 - *kOfxPropTime* the effect time at which changed occurred
 - *kOfxImageEffectPropRenderScale* the render scale applied to any image fetched
- **outArgs** – is redundant and is set to NULL

Pre

- *kOfxActionCreateInstance* has been called on the instance handle,

Returns

- *kOfxStatOK* the action was trapped
- *kOfxStatReplyDefault* the action was not trapped

Typedefs

```
typedef struct OfxInteract *OfxInteractHandle
```

Blind declaration of an OFX interactive gui.

```
typedef struct OfxInteractSuiteV1 OfxInteractSuiteV1
```

OFX suite that allows an effect to interact with an OpenGL window so as to provide custom interfaces.

file **ofxKeySyms.h**

Defines

kOfxPropKeySym

Property used to indicate which a key on the keyboard or a button on a button device has been pressed.

- Type - int X 1
- Property Set - an read only in argument for the actions *kOfxInteractActionKeyDown*, *kOfxInteractActionKeyUp* and *kOfxInteractActionKeyRepeat*.
- Valid Values - one of any specified by #defines in the file *ofxKeySyms.h*.

This property represents a raw key press, it does not represent the ‘character value’ of the key.

This property is associated with a *kOfxPropKeyString* property, which encodes the UTF8 value for the keypress/button press. Some keys (for example arrow keys) have no UTF8 equivalent.

Some keys, especially on non-english language systems, may have a UTF8 value, but *not* a keysym values, in these cases, the keysym will have a value of *kOfxKey_Unknown*, but the *kOfxPropKeyString* property will still be set with the UTF8 value.

kOfxPropKeyString

This property encodes a single keypresses that generates a unicode code point. The value is stored as a UTF8 string.

- Type - C string X 1, UTF8
- Property Set - an read only in argument for the actions *kOfxInteractActionKeyDown*, *kOfxInteractActionKeyUp* and *kOfxInteractActionKeyRepeat*.
- Valid Values - a UTF8 string representing a single character, or the empty string.

This property represents the UTF8 encode value of a single key press by a user in an OFX interact.

This property is associated with a *kOfxPropKeySym* which represents an integer value for the key press. Some keys (for example arrow keys) have no UTF8 equivalent, in which case this is set to the empty string “”, and the associate *kOfxPropKeySym* is set to the equivalent raw key press.

Some keys, especially on non-english language systems, may have a UTF8 value, but *not* a keysym values, in these cases, the keysym will have a value of *kOfxKey_Unknown*, but the *kOfxPropKeyString* property will still be set with the UTF8 value.

kOfxKey_Unknown

kOfxKey_BackSpace

kOfxKey_Tab

kOfxKey_Linefeed

kOfxKey_Clear

kOfxKey_Return

kOfxKey_Pause

kOfxKey_Scroll_Lock

kOfxKey_Sys_Req

kOfxKey_Escape

kOfxKey_Delete

kOfxKey_Multi_key

kOfxKey_SingleCandidate

kOfxKey_MultipleCandidate

kOfxKey_PreviousCandidate

kOfxKey_Kanji

kOfxKey_Muhenkan

kOfxKey_Henkan_Mode

kOfxKey_Henkan

kOfxKey_Romaji

kOfxKey_Hiragana

kOfxKey_Katakana

kOfxKey_Hiragana_Katakana

kOfxKey_Zenkaku

kOfxKey_Hankaku

kOfxKey_Zenkaku_Hankaku

kOfxKey_Touroku

kOfxKey_Massyo

kOfxKey_Kana_Lock

kOfxKey_Kana_Shift

kOfxKey_Eisu_Shift

kOfxKey_Eisu_toggle

kOfxKey_Zen_Koho

kOfxKey_Mae_Koho

kOfxKey_Home

kOfxKey_Left

kOfxKey_Up

kOfxKey_Right

kOfxKey_Down

kOfxKey_Prior

kOfxKey_Page_Up

kOfxKey_Next

kOfxKey_Page_Down

kOfxKey_End

kOfxKey_Begin

kOfxKey_Select

kOfxKey_Print

kOfxKey_Execute

kOfxKey_Insert

kOfxKey_Undo

kOfxKey_Redo

kOfxKey_Menu

kOfxKey_Find

kOfxKey_Cancel

kOfxKey_Help

kOfxKey_Break

kOfxKey_Mode_switch

kOfxKey_script_switch

kOfxKey_Num_Lock

kOfxKey_KP_Space

kOfxKey_KP_Tab

kOfxKey_KP_Enter

kOfxKey_KP_F1

kOfxKey_KP_F2

kOfxKey_KP_F3

kOfxKey_KP_F4

kOfxKey_KP_Home

kOfxKey_KP_Left

kOfxKey_KP_Up

kOfxKey_KP_Right

kOfxKey_KP_Down

kOfxKey_KP_Prior

kOfxKey_KP_Page_Up

kOfxKey_KP_Next

kOfxKey_KP_Page_Down

kOfxKey_KP_End

kOfxKey_KP_Begin

kOfxKey_KP_Insert

kOfxKey_KP_Delete

kOfxKey_KP_Equal

kOfxKey_KP_Multiply

kOfxKey_KP_Add

kOfxKey_KP_Separator

kOfxKey_KP_Subtract

kOfxKey_KP_Decimal

kOfxKey_KP_Divide

kOfxKey_KP_0

kOfxKey_KP_1

kOfxKey_KP_2

kOfxKey_KP_3

kOfxKey_KP_4

kOfxKey_KP_5

kOfxKey_KP_6

kOfxKey_KP_7

kOfxKey_KP_8

kOfxKey_KP_9

kOfxKey_F1

kOfxKey_F2

kOfxKey_F3

kOfxKey_F4

kOfxKey_F5

kOfxKey_F6

kOfxKey_F7

kOfxKey_F8

kOfxKey_F9

kOfxKey_F10

kOfxKey_F11

kOfxKey_L1

kOfxKey_F12

kOfxKey_L2

kOfxKey_F13

kOfxKey_L3

kOfxKey_F14

kOfxKey_L4

kOfxKey_F15

kOfxKey_L5

kOfxKey_F16

kOfxKey_L6

kOfxKey_F17

kOfxKey_L7

kOfxKey_F18

kOfxKey_L8

kOfxKey_F19

kOfxKey_L9

kOfxKey_F20

kOfxKey_L10

kOfxKey_F21

kOfxKey_R1

kOfxKey_F22

kOfxKey_R2

kOfxKey_F23

kOfxKey_R3

kOfxKey_F24

kOfxKey_R4

kOfxKey_F25

kOfxKey_R5

kOfxKey_F26

kOfxKey_R6

kOfxKey_F27

kOfxKey_R7

kOfxKey_F28

kOfxKey_R8

kOfxKey_F29

kOfxKey_R9

kOfxKey_F30

kOfxKey_R10

kOfxKey_F31

kOfxKey_R11

kOfxKey_F32

kOfxKey_R12

kOfxKey_F33

kOfxKey_R13

kOfxKey_F34

kOfxKey_R14

kOfxKey_F35

kOfxKey_R15

kOfxKey_Shift_L

kOfxKey_Shift_R

kOfxKey_Control_L

kOfxKey_Control_R

kOfxKey_Caps_Lock

kOfxKey_Shift_Lock

kOfxKey_Meta_L

kOfxKey_Meta_R

kOfxKey_Alt_L

kOfxKey_Alt_R

kOfxKey_Super_L

kOfxKey_Super_R

kOfxKey_Hyper_L

kOfxKey_Hyper_R

kOfxKey_space

kOfxKey_exclam

kOfxKey_quotedbl

kOfxKey_numbersign

kOfxKey_dollar

kOfxKey_percent

kOfxKey_ampersand

kOfxKey_apostrophe

kOfxKey_quoteright

kOfxKey_parenleft

kOfxKey_parenright

kOfxKey_asterisk

kOfxKey_plus

kOfxKey_comma

kOfxKey_minus

kOfxKey_period

kOfxKey_slash

kOfxKey_0

kOfxKey_1

kOfxKey_2

kOfxKey_3

kOfxKey_4

kOfxKey_5

kOfxKey_6

kOfxKey_7

kOfxKey_8

kOfxKey_9

kOfxKey_colon

kOfxKey_semicolon

kOfxKey_less

kOfxKey_equal

kOfxKey_greater

kOfxKey_question

kOfxKey_at

kOfxKey_A

kOfxKey_B

kOfxKey_C

kOfxKey_D

kOfxKey_E

kOfxKey_F

kOfxKey_G

kOfxKey_H

kOfxKey_I

kOfxKey_J

kOfxKey_K

kOfxKey_L

kOfxKey_M

kOfxKey_N

kOfxKey_O

kOfxKey_P

kOfxKey_Q

kOfxKey_R

kOfxKey_S

kOfxKey_T

kOfxKey_U

kOfxKey_V

kOfxKey_W

kOfxKey_X

kOfxKey_Y

kOfxKey_Z

kOfxKey_bracketleft

kOfxKey_backslash

kOfxKey_bracketright

kOfxKey_asciicircum

kOfxKey_underscore

kOfxKey_grave

kOfxKey_quoteleft

kOfxKey_a

kOfxKey_b

kOfxKey_c

kOfxKey_d

kOfxKey_e

kOfxKey_f

kOfxKey_g

kOfxKey_h

kOfxKey_i

kOfxKey_j

kOfxKey_k

kOfxKey_l

kOfxKey_m

kOfxKey_n

kOfxKey_o

kOfxKey_p

kOfxKey_q

kOfxKey_r

kOfxKey_s

kOfxKey_t

kOfxKey_u

kOfxKey_v

kOfxKey_w

kOfxKey_x

kOfxKey_y

kOfxKey_z

kOfxKey_braceleft

kOfxKey_bar

kOfxKey_braceright

kOfxKey_asciitilde

kOfxKey_nobreakspace

kOfxKey_exclamdown

kOfxKey_cent

kOfxKey_sterling

kOfxKey_currency

kOfxKey_yen

kOfxKey_brokenbar

kOfxKey_section

kOfxKey_diaeresis

kOfxKey_copyright

kOfxKey_ordfeminine

k0fxKey_guillemotleft

k0fxKey_notsign

k0fxKey_hyphen

k0fxKey_registered

k0fxKey_macron

k0fxKey_degree

k0fxKey_plusminus

k0fxKey_twosuperior

k0fxKey_threesuperior

k0fxKey_acute

k0fxKey_mu

k0fxKey_paragraph

k0fxKey_periodcentered

k0fxKey_cedilla

k0fxKey_onesuperior

k0fxKey_masculine

k0fxKey_guillemotright

k0fxKey_onequarter

k0fxKey_onehalf

k0fxKey_threequarters

k0fxKey_questiondown

kOfxKey_Agrave

kOfxKey_Aacute

kOfxKey_Acircumflex

kOfxKey_Atilde

kOfxKey_Adiaeresis

kOfxKey_Aring

kOfxKey_AE

kOfxKey_Ccedilla

kOfxKey_Egrave

kOfxKey_Eacute

kOfxKey_Ecircumflex

kOfxKey_Ediaeresis

kOfxKey_Igrave

kOfxKey_Iacute

kOfxKey_Icircumflex

kOfxKey_Idiaeresis

kOfxKey_ETH

kOfxKey_Eth

kOfxKey_Ntilde

kOfxKey_Ograve

kOfxKey_Oacute

kOfxKey_0circumflex

kOfxKey_0tilde

kOfxKey_0diaeresis

kOfxKey_multiply

kOfxKey_0oblique

kOfxKey_Ugrave

kOfxKey_Uacute

kOfxKey_Ucircumflex

kOfxKey_Udiaeresis

kOfxKey_Yacute

kOfxKey_THORN

kOfxKey_ssharp

kOfxKey_agrave

kOfxKey_aacute

kOfxKey_acircumflex

kOfxKey_atilde

kOfxKey_adiaeresis

kOfxKey_aring

kOfxKey_ae

kOfxKey_ccedilla

kOfxKey_egrave

k0fxKey_eacute

k0fxKey_ecircumflex

k0fxKey_ediaeresis

k0fxKey_igrave

k0fxKey_iacute

k0fxKey_icircumflex

k0fxKey_idiaeresis

k0fxKey_eth

k0fxKey_ntilde

k0fxKey_ograve

k0fxKey_oacute

k0fxKey_ocircumflex

k0fxKey_otilde

k0fxKey_odiaeresis

k0fxKey_division

k0fxKey_oslash

k0fxKey_ugrave

k0fxKey_uacute

k0fxKey_ucircumflex

k0fxKey_udiaeresis

k0fxKey_yacute

kOfxKey_thorn

kOfxKey_ydiaeresis

file **ofxMemory.h**

This file contains the API for general purpose memory allocation from a host.

Defines

kOfxMemorySuite

Typedefs

typedef struct *OfxMemorySuiteV1* **OfxMemorySuiteV1**

The OFX suite that implements general purpose memory management.

Use this suite for ordinary memory management functions, where you would normally use malloc/free or new/delete on ordinary objects.

For images, you should use the memory allocation functions in the image effect suite, as many hosts have specific image memory pools.

Note: C++ plugin developers will need to redefine new and delete as skins on top of this suite.

file **ofxMessage.h**

#include "ofxCore.h" This file contains the Host API for end user message communication.

Defines

kOfxMessageSuite

kOfxMessageFatal

String used to type fatal error messages.

Fatal error messages should only be posted by a plugin when it can no longer continue operation.

kOfxMessageError

String used to type error messages.

Ordinary error messages should be posted when there is an error in operation that is recoverable by user intervention.

kOfxMessageWarning

String used to type warning messages.

Warnings indicate states that allow for operations to proceed, but are not necessarily optimal.

kOfxMessageMessage

String used to type simple ordinary messages.

Ordinary messages simply convey information from the plugin directly to the user.

kOfxMessageLog

String used to type log messages.

Log messages are written out to a log and not to the end user.

kOfxMessageQuestion

String used to type yes/no messages.

The host is to enter a modal state which waits for the user to respond yes or no. The *OfxMessageSuiteV1::message* function which posted the message will only return after the user responds. When asking a question, the *OfxStatus* code returned by the message function will be,

- *kOfxStatReplyYes* - if the user replied ‘yes’ to the question
- *kOfxStatReplyNo* - if the user replied ‘no’ to the question
- some error code if an error was encountered

It is an error to post a question message if the plugin is not in an interactive session.

Typedefs

typedef struct *OfxMessageSuiteV1* **OfxMessageSuiteV1**

The OFX suite that allows a plug-in to pass messages back to a user. The V2 suite extends on this in a backwards compatible manner.

typedef struct *OfxMessageSuiteV2* **OfxMessageSuiteV2**

The OFX suite that allows a plug-in to pass messages back to a user.

This extends *OfxMessageSuiteV1*, and should be considered a replacement to version 1.

Note that this suite has been extended in backwards compatible manner, so that a host can return this struct for both V1 and V2.

file **ofxMultiThread.h**

#include “*ofxCore.h*” This file contains the Host Suite for threading

Defines

kOfxMultiThreadSuite

Typedefs

typedef struct OfxMutex ***OfxMutexHandle**

Mutex blind data handle.

void() **OfxThreadFunctionV1** (unsigned int threadIndex, unsigned int threadMax, void *customArg)

The function type to be passed to the multi threading routines.

- **threadIndex** unique index of this thread, will be between 0 and **threadMax**
- **threadMax** to total number of threads executing this function
- **customArg** the argument passed into **multiThread**

A function of this type is passed to *OfxMultiThreadSuiteV1::multiThread* to be launched in multiple threads.

typedef struct *OfxMultiThreadSuiteV1* **OfxMultiThreadSuiteV1**

OFX suite that provides simple SMP style multi-processing.

file **ofxOld.h**

Defines

kOfxImageComponentYUVA

String to label images with YUVA components —ofxImageEffects.h.

Deprecated:

- removed in v1.4. Note, this has been deprecated in v1.3

kOfxImageEffectPropInAnalysis

Indicates whether an effect is performing an analysis pass. —ofxImageEffects.h.

- Type - int X 1
- Property Set - plugin instance (read/write)
- Default - to 0
- Valid Values - This must be one of 0 or 1

Deprecated:

- This feature has been deprecated - officially commented out v1.4.

kOfxInteractPropViewportSize

The size of an interact's OpenGL viewport — *ofxInteract.h*.

- Type - int X 2
- Property Set - read only property on the interact instance and in argument to all the interact actions.

Deprecated:

- V1.3: This property is the redundant and its use will be deprecated in future releases. V1.4: Removed

kOfxParamDoubleTypeNormalisedX

value for the *kOfxParamPropDoubleType* property, indicating a size normalised to the X dimension. See *kOfxParamPropDoubleType*. — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of ::OfxParamDoubleTypeX V1.4: Removed

kOfxParamDoubleTypeNormalisedY

value for the *kOfxParamPropDoubleType* property, indicating a size normalised to the Y dimension. See *kOfxParamPropDoubleType*. — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of ::OfxParamDoubleTypeY V1.4: Removed

kOfxParamDoubleTypeNormalisedXAbsolute

value for the *kOfxParamPropDoubleType* property, indicating an absolute position normalised to the X dimension. See *kOfxParamPropDoubleType*. — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of ::OfxParamDoubleTypeXAbsolute V1.4: Removed

kOfxParamDoubleTypeNormalisedYAbsolute

value for the *kOfxParamPropDoubleType* property, indicating an absolute position normalised to the Y dimension. See *kOfxParamPropDoubleType*. — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of ::OfxParamDoubleTypeYAbsolute V1.4: Removed

kOfxParamDoubleTypeNormalisedXY

value for the *kOfxParamPropDoubleType* property, indicating normalisation to the X and Y dimension for 2D params. See *kOfxParamPropDoubleType*. — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of ::OfxParamDoubleTypeXY V1.4: Removed

kOfxParamDoubleTypeNormalisedXYAbsolute

value for the *kOfxParamPropDoubleType* property, indicating normalisation to the X and Y dimension for a 2D param that can be interpreted as an absolute spatial position. See *kOfxParamPropDoubleType*.
 — *ofxParam.h*

Deprecated:

- V1.3: Deprecated in favour of *kOfxParamDoubleTypeXYAbsolute* V1.4: Removed

Typedefs

typedef struct *OfxYUVAColourB* **OfxYUVAColourB**

Defines an 8 bit per component YUVA pixel — *ofxPixels.h* Deprecated in 1.3, removed in 1.4.

typedef struct *OfxYUVAColourS* **OfxYUVAColourS**

Defines an 16 bit per component YUVA pixel — *ofxPixels.h*.

Deprecated:

- Deprecated in 1.3, removed in 1.4

typedef struct *OfxYUVAColourF* **OfxYUVAColourF**

Defines an floating point component YUVA pixel — *ofxPixels.h*.

Deprecated:

- Deprecated in 1.3, removed in 1.4

file **ofxOpenGLRender.h**

Defines

_ofxOpenGLRender_h_

file **ofxParam.h**

#include "ofxCore.h" *#include "ofxProperty.h"* This header contains the suite definition to manipulate host side parameters.

For more details go see ParametersPage

Defines

kOfxParameterSuite

string value to the *kOfxPropType* property for all parameters

kOfxTypeParameter

string value on the *kOfxPropType* property for all parameter definitions (ie: the handle returned in describe)

kOfxTypeParameterInstance

string value on the *kOfxPropType* property for all parameter instances

kOfxParamTypeInteger

String to identify a param as a single valued integer.

kOfxParamTypeDouble

String to identify a param as a Single valued floating point parameter

kOfxParamTypeBoolean

String to identify a param as a Single valued boolean parameter.

kOfxParamTypeChoice

String to identify a param as a Single valued, ‘one-of-many’ parameter.

kOfxParamTypeStrChoice

String to identify a param as a string-valued ‘one-of-many’ parameter.

Since

Version 1.5

kOfxParamTypeRGBA

String to identify a param as a Red, Green, Blue and Alpha colour parameter.

kOfxParamTypeRGB

String to identify a param as a Red, Green and Blue colour parameter.

kOfxParamTypeDouble2D

String to identify a param as a Two dimensional floating point parameter.

kOfxParamTypeInteger2D

String to identify a param as a Two dimensional integer point parameter.

kOfxParamTypeDouble3D

String to identify a param as a Three dimensional floating point parameter.

kOfxParamTypeInteger3D

String to identify a param as a Three dimensional integer parameter.

kOfxParamTypeString

String to identify a param as a String (UTF8) parameter.

kOfxParamTypeCustom

String to identify a param as a Plug-in defined parameter.

kOfxParamTypeGroup

String to identify a param as a Grouping parameter.

kOfxParamTypePage

String to identify a param as a page parameter.

kOfxParamTypePushButton

String to identify a param as a PushButton parameter.

kOfxParamHostPropSupportsCustomAnimation

Indicates if the host supports animation of custom parameters.

- Type - int X 1
- Property Set - host descriptor (read only)
- Value Values - 0 or 1

kOfxParamHostPropSupportsStringAnimation

Indicates if the host supports animation of string params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

kOfxParamHostPropSupportsBooleanAnimation

Indicates if the host supports animation of boolean params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

kOfxParamHostPropSupportsChoiceAnimation

Indicates if the host supports animation of choice params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

kOfxParamHostPropSupportsCustomInteract

Indicates if the host supports custom interacts for parameters.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

Currently custom interacts for parameters can only be drawn using OpenGL. APIs will be added later to support using the new Draw Suite.

kOfxParamHostPropMaxParameters

Indicates the maximum numbers of parameters available on the host.

- Type - int X 1
- Property Set - host descriptor (read only)

If set to -1 it implies unlimited number of parameters.

kOfxParamHostPropMaxPages

Indicates the maximum number of parameter pages.

- Type - int X 1
- Property Set - host descriptor (read only)

If there is no limit to the number of pages on a host, set this to -1.

Hosts that do not support paged parameter layout should set this to zero.

kOfxParamHostPropPageRowColumnCount

This indicates the number of parameter rows and columns on a page.

- Type - int X 2
- Property Set - host descriptor (read only)

If the host has supports paged parameter layout, used dimension 0 as the number of columns per page and dimension 1 as the number of rows per page.

kOfxParamPageSkipRow

Pseudo parameter name used to skip a row in a page layout.

Passed as a value to the *kOfxParamPropPageChild* property.

See *ParametersInterfacesPagedLayouts* for more details.

kOfxParamPageSkipColumn

Pseudo parameter name used to skip a row in a page layout.

Passed as a value to the *kOfxParamPropPageChild* property.

See ParametersInterfacesPagedLayouts for more details.

kOfxParamPropInteractV1

Overrides the parameter's standard user interface with the given interact.

- Type - pointer X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - NULL
- Valid Values - must point to a OfxPluginEntryPoint

If set, the parameter's normal interface is replaced completely by the interact gui.

Currently custom interacts for parameters can only be drawn using OpenGL. APIs will be added later to support using the new Draw Suite.

kOfxParamPropInteractSize

The size of a parameter instance's custom interface in screen pixels.

- Type - double x 2
- Property Set - plugin parameter instance (read only)

This is set by a host to indicate the current size of a custom interface if the plug-in has one. If not this is set to (0,0).

kOfxParamPropInteractSizeAspect

The preferred aspect ratio of a parameter's custom interface.

- Type - double x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1.0
- Valid Values - greater than or equal to 0.0

If set to anything other than 0.0, the custom interface for this parameter will be of a size with this aspect ratio (x size/y size).

kOfxParamPropInteractMinimumSize

The minimum size of a parameter's custom interface, in screen pixels.

- Type - double x 2
- Property Set - plugin parameter descriptor (read/write) and instance (read only)

- Default - 10,10
- Valid Values - greater than (0, 0)

Any custom interface will not be less than this size.

kOfxParamPropInteractPreferredSize

The preferred size of a parameter's custom interface.

- Type - int x 2
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 10,10
- Valid Values - greater than (0, 0)

A host should attempt to set a parameter's custom interface on a parameter to be this size if possible, otherwise it will be of *kOfxParamPropInteractSizeAspect* aspect but larger than *kOfxParamPropInteractMinimumSize*.

kOfxParamPropType

The type of a parameter.

- Type - C string X 1
- Property Set - plugin parameter descriptor (read only) and instance (read only)

This string will be set to the type that the parameter was create with.

kOfxParamPropAnimates

Flags whether a parameter can animate.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

A plug-in uses this property to indicate if a parameter is able to animate.

kOfxParamPropCanUndo

Flags whether changes to a parameter should be put on the undo/redo stack.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

kOfxPropParamSetNeedsSyncing

States whether the plugin needs to resync its private data.

- Type - int X 1
- Property Set - param set instance (read/write)
- Default - 0
- Valid Values -
 - 0 - no need to sync
 - 1 - paramset is not synced

The plugin should set this flag to true whenever any internal state has not been flushed to the set of params.

The host will examine this property each time it does a copy or save operation on the instance. If it is set to 1, the host will call SyncPrivateData and then set it to zero before doing the copy/save. If it is set to 0, the host will assume that the param data correctly represents the private state, and will not call SyncPrivateData before copying/saving. If this property is not set, the host will always call SyncPrivateData before copying or saving the effect (as if the property were set to 1 — but the host will not create or modify the property).

kOfxParamPropIsAnimating

Flags whether a parameter is currently animating.

- Type - int x 1
- Property Set - plugin parameter instance (read only)
- Valid Values - 0 or 1

Set by a host on a parameter instance to indicate if the parameter has a non-constant value set on it. This can be as a consequence of animation or of scripting modifying the value, or of a parameter being connected to an expression in the host.

kOfxParamPropPluginMayWrite

Flags whether the plugin will attempt to set the value of a parameter in some callback or analysis pass.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 0
- Valid Values - 0 or 1

This is used to tell the host whether the plug-in is going to attempt to set the value of the parameter.

Deprecated:

- v1.4: deprecated - to be removed in 1.5

kOfxParamPropPersistant

Flags whether the value of a parameter should persist.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

This is used to tell the host whether the value of the parameter is important and should be save in any description of the plug-in.

kOfxParamPropEvaluateOnChange

Flags whether changing a parameter's value forces an evaluation (ie: render),.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write only)
- Default - 1
- Valid Values - 0 or 1

This is used to indicate if the value of a parameter has any affect on an effect's output, eg: the parameter may be purely for GUI purposes, and so changing its value should not trigger a re-render.

kOfxParamPropSecret

Flags whether a parameter should be exposed to a user,.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write)
- Default - 0
- Valid Values - 0 or 1

If secret, a parameter is not exposed to a user in any interface, but should otherwise behave as a normal parameter.

Secret params are typically used to hide important state detail that would otherwise be unintelligible to a user, for example the result of a statical analysis that might need many parameters to store.

kOfxParamPropScriptName

The value to be used as the id of the parameter in a host scripting language.

- Type - ASCII C string X 1,
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - the unique name the parameter was created with.
- Valid Values - ASCII string unique to all parameters in the plug-in.

Many hosts have a scripting language that they use to set values of parameters and more. If so, this is the name of a parameter in such scripts.

kOfxParamPropCacheInvalidation

Specifies how modifying the value of a param will affect any output of an effect over time.

- Type - C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - *kOfxParamInvalidateValueChange*
- Valid Values - This must be one of
 - *kOfxParamInvalidateValueChange*
 - *kOfxParamInvalidateValueChangeToEnd*
 - *kOfxParamInvalidateAll*

Imagine an effect with an animating parameter in a host that caches rendered output. Think of the what happens when you add a new key frame. -If the parameter represents something like an absolute position, the cache will only need to be invalidated for the range of frames that keyframe affects.

- If the parameter represents something like a speed which is integrated, the cache will be invalidated from the keyframe until the end of the clip.
- There are potentially other situations where the entire cache will need to be invalidated (though I can't think of one off the top of my head).

kOfxParamInvalidateValueChange

Used as a value for the *kOfxParamPropCacheInvalidation* property.

kOfxParamInvalidateValueChangeToEnd

Used as a value for the *kOfxParamPropCacheInvalidation* property.

kOfxParamInvalidateAll

Used as a value for the *kOfxParamPropCacheInvalidation* property.

kOfxParamPropHint

A hint to the user as to how the parameter is to be used.

- Type - UTF8 C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - ""

kOfxParamPropDefault

The default value of a parameter.

- Type - The type is dependant on the parameter type as is the dimension.
- Property Set - plugin parameter descriptor (read/write) and instance (read/write only),

- Default - 0 cast to the relevant type (or "" for strings and custom parameters)

The exact type and dimension is dependant on the type of the parameter. These are...

- *kOfxParamTypeInteger* - integer property of one dimension
- *kOfxParamTypeDouble* - double property of one dimension
- *kOfxParamTypeBoolean* - integer property of one dimension
- *kOfxParamTypeChoice* - integer property of one dimension
- *kOfxParamTypeStrChoice* - string property of one dimension
- *kOfxParamTypeRGBA* - double property of four dimensions
- *kOfxParamTypeRGB* - double property of three dimensions
- *kOfxParamTypeDouble2D* - double property of two dimensions
- *kOfxParamTypeInteger2D* - integer property of two dimensions
- *kOfxParamTypeDouble3D* - double property of three dimensions
- *kOfxParamTypeInteger3D* - integer property of three dimensions
- *kOfxParamTypeString* - string property of one dimension
- *kOfxParamTypeCustom* - string property of one dimension
- *kOfxParamTypeGroup* - does not have this property
- *kOfxParamTypePage* - does not have this property
- *kOfxParamTypePushButton* - does not have this property

kOfxParamPropDoubleType

Describes how the double parameter should be interpreted by a host.

- Type - C string X 1
- Default - *kOfxParamDoubleTypePlain*
- Property Set - 1D, 2D and 3D float plugin parameter descriptor (read/write) and instance (read only),
- Valid Values -This must be one of
 - *kOfxParamDoubleTypePlain* - parameter has no special interpretation,
 - *kOfxParamDoubleTypeAngle* - parameter is to be interpreted as an angle,
 - *kOfxParamDoubleTypeScale* - parameter is to be interpreted as a scale factor,
 - *kOfxParamDoubleTypeTime* - parameter represents a time value (1D only),
 - *kOfxParamDoubleTypeAbsoluteTime* - parameter represents an absolute time value (1D only),
 - *kOfxParamDoubleTypeX* - size wrt to the project's X dimension (1D only), in canonical coordinates,
 - *kOfxParamDoubleTypeXAbsolute* - absolute position on the X axis (1D only), in canonical coordinates,
 - *kOfxParamDoubleTypeY* - size wrt to the project's Y dimension(1D only), in canonical coordinates,

- *kOfxParamDoubleTypeYAbsolute* - absolute position on the Y axis (1D only), in canonical coordinates,
- *kOfxParamDoubleTypeXY* - size in 2D (2D only), in canonical coordinates,
- *kOfxParamDoubleTypeXYAbsolute* - an absolute position on the image plane, in canonical coordinates.

Double parameters can be interpreted in several different ways, this property tells the host how to do so and thus gives hints as to the interface of the parameter.

kOfxParamDoubleTypePlain

value for the *kOfxParamPropDoubleType* property, indicating the parameter has no special interpretation and should be interpreted as a raw numeric value.

kOfxParamDoubleTypeScale

value for the *kOfxParamPropDoubleType* property, indicating the parameter is to be interpreted as a scale factor. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeAngle

value for the *kOfxParamDoubleTypeAngle* property, indicating the parameter is to be interpreted as an angle. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeTime

value for the *kOfxParamDoubleTypeAngle* property, indicating the parameter is to be interpreted as a time. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeAbsoluteTime

value for the *kOfxParamDoubleTypeAngle* property, indicating the parameter is to be interpreted as an absolute time from the start of the effect. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeX

value for the *kOfxParamPropDoubleType* property, indicating a size in canonical coords in the X dimension. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeY

value for the *kOfxParamPropDoubleType* property, indicating a size in canonical coords in the Y dimension. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeXAbsolute

value for the *kOfxParamPropDoubleType* property, indicating an absolute position in canonical coords in the X dimension. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeYAbsolute

value for the *kOfxParamPropDoubleType* property, indicating an absolute position in canonical coords in the Y dimension. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeXY

value for the *kOfxParamPropDoubleType* property, indicating a 2D size in canonical coords. See *kOfxParamPropDoubleType*.

kOfxParamDoubleTypeXYAbsolute

value for the *kOfxParamPropDoubleType* property, indicating a 2D position in canonical coords. See *kOfxParamPropDoubleType*.

kOfxParamPropDefaultCoordinateSystem

Describes in which coordinate system a spatial double parameter's default value is specified.

- Type - C string X 1
- Default - kOfxParamCoordinatesCanonical
- Property Set - Non normalised spatial double parameters, ie: any double param who's *kOfxParamPropDoubleType* is set to one of...
 - kOfxParamDoubleTypeX
 - kOfxParamDoubleTypeXAbsolute
 - kOfxParamDoubleTypeY
 - kOfxParamDoubleTypeYAbsolute
 - kOfxParamDoubleTypeXY
 - kOfxParamDoubleTypeXYAbsolute
- Valid Values - This must be one of
 - kOfxParamCoordinatesCanonical - the default is in canonical coords
 - kOfxParamCoordinatesNormalised - the default is in normalised coordinates

This allows a spatial param to specify what its default is, so by saying normalised and “0.5” it would be in the ‘middle’, by saying canonical and 100 it would be at value 100 independent of the size of the image being applied to.

kOfxParamCoordinatesCanonical

Define the canonical coordinate system.

kOfxParamCoordinatesNormalised

Define the normalised coordinate system.

kOfxParamPropHasHostOverlayHandle

A flag to indicate if there is a host overlay UI handle for the given parameter.

- Type - int x 1
- Property Set - plugin parameter descriptor (read only)
- Valid Values - 0 or 1

If set to 1, then the host is flagging that there is some sort of native user overlay interface handle available for the given parameter.

kOfxParamPropUseHostOverlayHandle

A flag to indicate that the host should use a native UI overlay handle for the given parameter.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write only) and instance (read only)
- Default - 0
- Valid Values - 0 or 1

If set to 1, then a plugin is flagging to the host that the host should use a native UI overlay handle for the given parameter. A plugin can use this to keep a native look and feel for parameter handles. A plugin can use *kOfxParamPropHasHostOverlayHandle* to see if handles are available on the given parameter.

kOfxParamPropShowTimeMarker

Enables the display of a time marker on the host's time line to indicate the value of the absolute time param.

- Type - int x 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write)
- Default - 0
- Valid Values - 0 or 1

If a double parameter is has *kOfxParamPropDoubleType* set to *kOfxParamDoubleTypeAbsoluteTime*, then this indicates whether any marker should be made visible on the host's time line.

kOfxPluginPropParamPageOrder

Sets the parameter pages and order of pages.

- Type - C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - ""
- Valid Values - the names of any page param in the plugin

This property sets the preferred order of parameter pages on a host. If this is never set, the preferred order is the order the parameters were declared in.

kOfxParamPropPageChild

The names of the parameters included in a page parameter.

- Type - C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - ""
- Valid Values - the names of any parameter that is not a group or page, as well as *kOfxParamPageSkipRow* and *kOfxParamPageSkipColumn*

This is a property on parameters of type *kOfxParamTypePage*, and tells the page what parameters it contains. The parameters are added to the page from the top left, filling in columns as we go. The two pseudo param names *kOfxParamPageSkipRow* and *kOfxParamPageSkipColumn* are used to control layout.

Note parameters can appear in more than one page.

kOfxParamPropParent

The name of a parameter's parent group.

- Type - C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - "", which implies the "root" of the hierarchy,
- Valid Values - the name of a parameter with type of *kOfxParamTypeGroup*

Hosts that have hierarchical layouts of their params use this to recursively group parameter.

By default parameters are added in order of declaration to the 'root' hierarchy. This property is used to reparent params to a predefined param of type *kOfxParamTypeGroup*.

kOfxParamPropGroupOpen

Whether the initial state of a group is open or closed in a hierarchical layout.

- Type - int X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only)
- Default - 1
- Valid Values - 0 or 1

This is a property on parameters of type *kOfxParamTypeGroup*, and tells the group whether it should be open or closed by default.

kOfxParamPropEnabled

Used to enable a parameter in the user interface.

- Type - int X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - 1
- Valid Values - 0 or 1

When set to 0 a user should not be able to modify the value of the parameter. Note that the plug-in itself can still change the value of a disabled parameter.

kOfxParamPropDataPtr

A private data pointer that the plug-in can store its own data behind.

- Type - pointer X 1

- Property Set - plugin parameter instance (read/write),
- Default - NULL

This data pointer is unique to each parameter instance, so two instances of the same parameter do not share the same data pointer. Use it to hang any needed private data structures.

kOfxParamPropChoiceOption

Set options of a choice parameter.

- Type - UTF8 C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the property is empty with no options set.

This property contains the set of options that will be presented to a user from a choice parameter. See ParametersChoice for more details.

kOfxParamPropChoiceOrder

Set values the host should store for a choice parameter.

- Type - int X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - Zero-based ordinal list of same length as OfxParamPropChoiceOption

This property specifies the order in which the options are presented. See “Choice Parameters” for more details. This property is optional; if not set, the host will present the options in their natural order.

This property is useful when changing order of choice param options, or adding new options in the middle, in a new version of the plugin.

```

Plugin v1:
Option = {"OptA", "OptB", "OptC"}
Order = {1, 2, 3}

Plugin v2:
// will be shown as OptA / OptB / NewOpt / OptC
Option = {"OptA", "OptB", "OptC", NewOpt"}
Order = {1, 2, 4, 3}

```

Note that this only affects the host UI’s display order; the project still stores the index of the selected option as always. Plugins should never reorder existing options if they desire backward compatibility.

Values may be arbitrary 32-bit integers. Behavior is undefined if the same value occurs twice in the list; plugins should not do that.

Since

Version 1.5

kOfxParamPropChoiceEnum

Set a enumeration string in a StrChoice (string-valued choice) parameter.

- Type - UTF8 C string X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the property is empty with no options set.

This property contains the set of enumeration strings stored by the host in the project corresponding to the options that will be presented to a user from a StrChoice parameter. See ParametersChoice for more details.

Since

Version 1.5

kOfxParamHostPropSupportsStrChoiceAnimation

Indicates if the host supports animation of string choice params.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

Since

Version 1.5

kOfxParamHostPropSupportsStrChoice

Indicates if the host supports the StrChoice param type.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values - 0 or 1

Since

Version 1.5

kOfxParamPropMin

The minimum value for a numeric parameter.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the smallest possible value corresponding to the parameter type (eg: INT_MIN for an integer, -DBL_MAX for a double parameter)

Setting this will also reset *kOfxParamPropDisplayMin*.

kOfxParamPropMax

The maximum value for a numeric parameter.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the largest possible value corresponding to the parameter type (eg: INT_MAX for an integer, DBL_MAX for a double parameter)

Setting this will also reset ::kOfxParamPropDisplayMax.

kOfxParamPropDisplayMin

The minimum value for a numeric parameter on any user interface.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the smallest possible value corresponding to the parameter type (eg: INT_MIN for an integer, -DBL_MAX for a double parameter)

If a user interface represents a parameter with a slider or similar, this should be the minimum bound on that slider.

kOfxParamPropDisplayMax

The maximum value for a numeric parameter on any user interface.

- Type - int or double X N
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - the largest possible value corresponding to the parameter type (eg: INT_MAX for an integer, DBL_MAX for a double parameter)

If a user interface represents a parameter with a slider or similar, this should be the maximum bound on that slider.

kOfxParamPropIncrement

The granularity of a slider used to represent a numeric parameter.

- Type - double X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - 1
- Valid Values - any greater than 0.

This value is always in canonical coordinates for double parameters that are normalised.

kOfxParamPropDigits

How many digits after a decimal point to display for a double param in a GUI.

- Type - int X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read/write),
- Default - 2
- Valid Values - any greater than 0.

This applies to double params of any dimension.

kOfxParamPropDimensionLabel

Label for individual dimensions on a multidimensional numeric parameter.

- Type - UTF8 C string X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - “x”, “y” and “z”
- Valid Values - any

Use this on 2D and 3D double and integer parameters to change the label on an individual dimension in any GUI for that parameter.

kOfxParamPropIsAutoKeying

Will a value change on the parameter add automatic keyframes.

- Type - int X 1
- Property Set - plugin parameter instance (read only),
- Valid Values - 0 or 1

This is set by the host simply to indicate the state of the property.

kOfxParamPropCustomInterpCallbackV1

A pointer to a custom parameter’s interpolation function.

- Type - pointer X 1
- Property Set - plugin parameter descriptor (read/write) and instance (read only),
- Default - NULL
- Valid Values - must point to a *OfxCustomParamInterpFuncV1*

It is an error not to set this property in a custom parameter during a plugin’s define call if the custom parameter declares itself to be an animating parameter.

kOfxParamPropStringMode

Used to indicate the type of a string parameter.

- Type - C string X 1
- Property Set - plugin string parameter descriptor (read/write) and instance (read only),
- Default - *kOfxParamStringIsSingleLine*
- Valid Values - This must be one of the following
 - *kOfxParamStringIsSingleLine*
 - *kOfxParamStringIsMultiLine*
 - *kOfxParamStringIsFilePath*
 - *kOfxParamStringIsDirectoryPath*
 - *kOfxParamStringIsLabel*
 - *kOfxParamStringIsRichTextFormat*

kOfxParamPropStringFilePathExists

Indicates string parameters of file or directory type need that file to exist already.

- Type - int X 1
- Property Set - plugin string parameter descriptor (read/write) and instance (read only),
- Default - 1
- Valid Values - 0 or 1

If set to 0, it implies the user can specify a new file name, not just a pre-existing one.

kOfxParamStringIsSingleLine

Used to set a string parameter to be single line, value to be passed to a *kOfxParamPropStringMode* property.

kOfxParamStringIsMultiLine

Used to set a string parameter to be multiple line, value to be passed to a *kOfxParamPropStringMode* property.

kOfxParamStringIsFilePath

Used to set a string parameter to be a file path, value to be passed to a *kOfxParamPropStringMode* property.

kOfxParamStringIsDirectoryPath

Used to set a string parameter to be a directory path, value to be passed to a *kOfxParamPropStringMode* property.

kOfxParamStringIsLabel

Use to set a string parameter to be a simple label, value to be passed to a *kOfxParamPropStringMode* property

kOfxParamStringIsRichTextFormat

String value on the *kOfxParamPropStringMode* property of a string parameter (added in 1.3)

kOfxParamPropCustomValue

Used by interpolating custom parameters to get and set interpolated values.

- Type - C string X 1 or 2

This property is on the *inArgs* property and *outArgs* property of a *OfxCustomParamInterpFuncV1* and in both cases contains the encoded value of a custom parameter. As an *inArgs* property it will have two values, being the two keyframes to interpolate. As an *outArgs* property it will have a single value and the plugin should fill this with the encoded interpolated value of the parameter.

kOfxParamPropInterpolationTime

Used by interpolating custom parameters to indicate the time a key occurs at.

- Type - double X 2
- Property Set - inArgs parameter of a *OfxCustomParamInterpFuncV1* (read only)

The two values indicate the absolute times the surrounding keyframes occur at. The keyframes are encoded in a *kOfxParamPropCustomValue* property.

kOfxParamPropInterpolationAmount

Property used by *OfxCustomParamInterpFuncV1* to indicate the amount of interpolation to perform.

- Type - double X 1
- Property Set - inArgs parameter of a *OfxCustomParamInterpFuncV1* (read only)
- Valid Values - from 0 to 1

This property indicates how far between the two *kOfxParamPropCustomValue* keys to interpolate.

Typedefs

```
typedef struct OfxParamStruct *OfxParamHandle
```

Blind declaration of an OFX param.

```
typedef struct OfxParamSetStruct *OfxParamSetHandle
```

Blind declaration of an OFX parameter set.

```
OfxStatus() OfxCustomParamInterpFuncV1 (OfxParamSetHandle instance,  
OfxPropertySetHandle inArgs, OfxPropertySetHandle outArgs)
```

Function prototype for custom parameter interpolation callback functions.

- *instance* the plugin instance that this parameter occurs in

- `inArgs` handle holding the following properties...
 - `kOfxPropName` - the name of the custom parameter to interpolate
 - `kOfxPropTime` - absolute time the interpolation is occurring at
 - `kOfxParamPropCustomValue` - string property that gives the value of the two keyframes to interpolate, in this case 2D
 - `kOfxParamPropInterpolationTime` - 2D double property that gives the time of the two keyframes we are interpolating
 - `kOfxParamPropInterpolationAmount` - 1D double property indicating how much to interpolate between the two keyframes
- `outArgs` handle holding the following properties to be set
 - `kOfxParamPropCustomValue` - the value of the interpolated custom parameter, in this case 1D

This function allows custom parameters to animate by performing interpolation between keys.

The plugin needs to parse the two strings encoding keyframes on either side of the time we need a value for. It should then interpolate a new value for it, encode it into a string and set the `kOfxParamPropCustomValue` property with this on the `outArgs` handle.

The interp value is a linear interpolation amount, however this may be derived from a cubic (or other) curve.

```
typedef struct OfxParameterSuiteV1 OfxParameterSuiteV1
```

The OFX suite used to define and manipulate user visible parameters.

file **ofxParametricParam.h**

`#include "ofxParam.h"` This header file defines the optional OFX extension to define and manipulate parametric parameters.

Defines

kOfxParametricParameterSuite

string value to the `kOfxPropType` property for all parameters

kOfxParamTypeParametric

String to identify a param as a single valued integer.

kOfxParamPropParametricDimension

The dimension of a parametric param.

- Type - int X 1
- Property Set - parametric param descriptor (read/write) and instance (read only)
- default - 1
- Value Values - greater than 0

This indicates the dimension of the parametric param.

kOfxParamPropParametricUIColor

The colour of parametric param curve interface in any UI.

- Type - double X N
- Property Set - parametric param descriptor (read/write) and instance (read only)
- default - unset,
- Value Values - three values for each dimension (see *kOfxParamPropParametricDimension*) being interpreted as R, G and B of the colour for each curve drawn in the UI.

This sets the colour of a parametric param curve drawn a host user interface. A colour triple is needed for each dimension of the oparametric param.

If not set, the host should generally draw these in white.

kOfxParamPropParametricInteractBackground

Interact entry point to draw the background of a parametric parameter.

- Type - pointer X 1
- Property Set - plug-in parametric parameter descriptor (read/write) and instance (read only),
- Default - NULL, which implies the host should draw its default background.

Defines a pointer to an interact which will be used to draw the background of a parametric parameter's user interface. None of the pen or keyboard actions can ever be called on the interact.

The OpenGL transform will be set so that it is an orthographic transform that maps directly to the 'parametric' space, so that 'x' represents the parametric position and 'y' represents the evaluated value.

kOfxParamHostPropSupportsParametricAnimation

Property on the host to indicate support for parametric parameter animation.

- Type - int X 1
- Property Set - host descriptor (read only)
- Valid Values
 - 0 indicating the host does not support animation of parametric params,
 - 1 indicating the host does support animation of parametric params,

kOfxParamPropParametricRange

Property to indicate the min and max range of the parametric input value.

- Type - double X 2
- Property Set - parameter descriptor (read/write only), and instance (read only)
- Default Value - (0, 1)
- Valid Values - any pair of numbers so that the first is less than the second.

This controls the min and max values that the parameter will be evaluated at.

Typedefs

typedef struct *OfxParametricParameterSuiteV1* **OfxParametricParameterSuiteV1**

The OFX suite used to define and manipulate ‘parametric’ parameters.

This is an optional suite.

Parametric parameters are in effect ‘functions’ a plug-in can ask a host to arbitrarily evaluate for some value ‘x’. A classic use case would be for constructing look-up tables, a plug-in would ask the host to evaluate one at multiple values from 0 to 1 and use that to fill an array.

A host would probably represent this to a user as a cubic curve in a standard curve editor interface, or possibly through scripting. The user would then use this to define the ‘shape’ of the parameter.

The evaluation of such params is not the same as animation, they are returning values based on some arbitrary argument orthogonal to time, so to evaluate such a param, you need to pass a parametric position and time.

Often, you would want such a parametric parameter to be multi-dimensional, for example, a colour look-up table might want three values, one for red, green and blue. Rather than declare three separate parametric parameters, it would be better to have one such parameter with multiple values in it.

The major complication with these parameters is how to allow a plug-in to set values, and defaults. The default default value of a parametric curve is to be an identity lookup. If a plugin wishes to set a different default value for a curve, it can use the suite to set key/value pairs on the *descriptor* of the param. When a new instance is made, it will have these curve values as a default.

file **ofxPixels.h**

Contains pixel struct definitions

Typedefs

typedef struct *OfxRGBAColourB* **OfxRGBAColourB**

Defines an 8 bit per component RGBA pixel.

typedef struct *OfxRGBAColourS* **OfxRGBAColourS**

Defines a 16 bit per component RGBA pixel.

typedef struct *OfxRGBAColourF* **OfxRGBAColourF**

Defines a floating point component RGBA pixel.

typedef struct *OfxRGBAColourD* **OfxRGBAColourD**

Defines a double precision floating point component RGBA pixel.

typedef struct *OfxRGBAColourB* **OfxRGBAColourB**

Defines an 8 bit per component RGB pixel.

typedef struct *OfxRGBColourS* **OfxRGBColourS**

Defines a 16 bit per component RGB pixel.

typedef struct *OfxRGBColourF* **OfxRGBColourF**

Defines a floating point component RGB pixel.

typedef struct *OfxRGBColourD* **OfxRGBColourD**

Defines a double precision floating point component RGB pixel.

file **ofxProgress.h**

Defines

kOfxProgressSuite

suite for displaying a progress bar

Typedefs

typedef struct *OfxProgressSuiteV1* **OfxProgressSuiteV1**

A suite that provides progress feedback from a plugin to an application.

A plugin instance can initiate, update and close a progress indicator with this suite.

This is an optional suite in the Image Effect API.

API V1.4: Amends the documentation of progress suite V1 so that it is expected that it can be raised in a modal manner and have a “cancel” button when invoked in instanceChanged. Plugins that perform analysis post an appropriate message, raise the progress monitor in a modal manner and should poll to see if processing has been aborted. Any cancellation should be handled gracefully by the plugin (eg: reset analysis parameters to default values), clear allocated memory...

Many hosts already operate as described above. kOfxStatReplyNo should be returned to the plugin during progressUpdate when the user presses cancel.

Suite V2: Adds an ID that can be looked up for internationalisation and so on. When a new version is introduced, because plug-ins need to support old versions, and plug-in’s new releases are not necessary in synch with hosts (or users don’t immediately update), best practice is to support the 2 suite versions. That is, the plugin should check if V2 exists; if not then check if V1 exists. This way a graceful transition is guaranteed. So plugin should fetchSuite passing 2, (OfxProgressSuiteV2*) fetchSuite(mHost->mHost->host, kOfxProgressSuite,2); and if no success pass (OfxProgressSuiteV1*) fetchSuite(mHost->mHost->host, kOfxProgressSuite,1);

typedef struct *OfxProgressSuiteV2* **OfxProgressSuiteV2**

file **ofxProperty.h**

#include “ofxCore.h” Contains the API for manipulating generic properties. For more details see PropertiesPage.

Defines**kOfxPropertySuite****Typedefs**typedef struct *OfxPropertySuiteV1* **OfxPropertySuiteV1**

The OFX suite used to access properties on OFX objects.

file **ofxTimeLine.h****Defines****kOfxTimeLineSuite**

Name of the time line suite.

Typedefstypedef struct *OfxTimeLineSuiteV1* **OfxTimeLineSuiteV1**

Suite to control timelines.

This suite is used to enquire and control a timeline associated with a plug-in instance.

This is an optional suite in the Image Effect API.

group **ActionsAll**

These are the actions passed to a plug-in's 'main' function

group **PropertiesAll**

These strings are used to identify properties within OFX, they are broken up by the host suite or API they relate to.

group **PropertiesGeneral**

These properties are general properties and apply to many objects across OFX

group **StatusCodes**

These strings are used to identify error states within ofx, they are returned by various host suite functions, as well as plug-in functions. The valid return codes for each function are documented with that function.

group **StatusCodesGeneral**

General status codes start at 1 and continue until 999

group **OpenGLRenderSuite**

StatusReturnValues

OfxStatus returns indicating that a OpenGL render error has occurred:

- If a plug-in returns *kOfxStatGLRenderFailed*, the host should retry the render with OpenGL rendering disabled.
- If a plug-in returns *kOfxStatGLOutOfMemory*, the host may choose to free resources on the GPU and retry the OpenGL render, rather than immediately falling back to CPU rendering.

kOfxStatGPUOutOfMemory

GPU render ran out of memory.

kOfxStatGLOutOfMemory

OpenGL render ran out of memory (same as *kOfxStatGPUOutOfMemory*)

kOfxStatGPURenderFailed

GPU render failed in a non-memory-related way.

kOfxStatGLRenderFailed

OpenGL render failed in a non-memory-related way (same as *kOfxStatGPURenderFailed*)

Defines

kOfxOpenGLRenderSuite

The name of the OpenGL render suite, used to fetch from a host via *OfxHost::fetchSuite*.

kOfxImageEffectPropOpenGLRenderSupported

Indicates whether a host or plug-in can support OpenGL accelerated rendering.

- Type - C string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only) - plug-in instance change (read/write)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - in which case the host or plug-in does not support OpenGL accelerated rendering
 - “true” - which means a host or plug-in can support OpenGL accelerated rendering, in the case of plug-ins this also means that it is capable of CPU based rendering in the absence of a GPU
 - “needed” - only for plug-ins, this means that an plug-in has to have OpenGL support, without which it cannot work.

V1.4: It is now expected from host reporting v1.4 that the plug-in can during instance change switch from true to false and false to true.

kOfxOpenGLPropPixelDepth

Indicates the bit depths supported by a plug-in during OpenGL renders.

This is analogous to *kOfxImageEffectPropSupportedPixelDepths*. When a plug-in sets this property, the host will try to provide buffers/textures in one of the supported formats. Additionally, the target buffers where the plug-in renders to will be set to one of the supported formats.

Unlike *kOfxImageEffectPropSupportedPixelDepths*, this property is optional. Shader-based effects might not really care about any format specifics when using OpenGL textures, so they can leave this unset and allow the host to decide the format.

- Type - string X N
- Property Set - plug-in descriptor (read only)
- Default - none set
- Valid Values - This must be one of
 - *kOfxBitDepthNone* (implying a clip is unconnected, not valid for an image)
 - *kOfxBitDepthByte*
 - *kOfxBitDepthShort*
 - *kOfxBitDepthHalf*
 - *kOfxBitDepthFloat*

kOfxImageEffectPropOpenGLEnabled

Indicates that a plug-in SHOULD use OpenGL acceleration in the current action.

When a plug-in and host have established they can both use OpenGL renders then when this property has been set the host expects the plug-in to render its result into the buffer it has setup before calling the render. The plug-in can then also safely use the ‘OfxImageEffectOpenGLRenderSuite’

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the plug-in cannot use the OpenGL suite
 - 1 indicates that the plug-in should render into the texture, and may use the OpenGL suite functions.

v1.4: *kOfxImageEffectPropOpenGLEnabled* should probably be checked in Instance Changed prior to try to read image via *clipLoadTexture*

Note: Once this property is set, the host and plug-in have agreed to use OpenGL, so the effect SHOULD access all its images through the OpenGL suite.

kOfxImageEffectPropOpenGLTextureIndex

Indicates the texture index of an image turned into an OpenGL texture by the host.

- Type - int X 1
- Property Set - texture handle returned by `OfxImageEffectOpenGLRenderSuiteV1::clipLoadTexture` (read only)

This value should be cast to a GLuint **and** used **as** the texture index when performing OpenGL texture operations.

The property set of the following actions should contain this property:

- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*

kOfxImageEffectPropOpenGLTextureTarget

Indicates the texture target enumerator of an image turned into an OpenGL texture by the host.

- Type - int X 1
- Property Set - texture handle returned by `OfxImageEffectOpenGLRenderSuiteV1::clipLoadTexture` (read only) This value should be cast to a GLenum and used as the texture target when performing OpenGL texture operations.

The property set of the following actions should contain this property:

- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*

kOfxActionOpenGLContextAttached

Action called when an effect has just been attached to an OpenGL context.

The purpose of this action is to allow a plug-in to set up any data it may need to do OpenGL rendering in an instance. For example...

- allocate a lookup table on a GPU,
- create an OpenCL or CUDA context that is bound to the host's OpenGL context so it can share buffers.

The plug-in will be responsible for deallocating any such shared resource in the `kOfxActionOpenGLContextDetached` action.

A host cannot call `kOfxActionOpenGLContextAttached` on the same instance without an intervening `kOfxActionOpenGLContextDetached`. A host can have a plug-in swap OpenGL contexts by issuing a `attach/detach` for the first context then another `attach` for the next context.

The arguments to the action are...

- `handle` handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- `inArgs` is redundant and set to NULL

- `outArgs` is redundant and set to NULL

A plug-in can return...

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored, but all was well anyway
- *kOfxStatErrMemory*, in which case this may be called again after a memory purge
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plug-in should to post a message if possible and the host should not attempt to run the plug-in in OpenGL render mode.

kOfxActionOpenGLContextDetached

Action called when an effect is about to be detached from an OpenGL context.

The purpose of this action is to allow a plug-in to deallocate any resource allocated in `kOfxActionOpenGLContextAttached` just before the host decouples a plug-in from an OpenGL context. The host must call this with the same OpenGL context active as it called with the corresponding `kOfxActionOpenGLContextAttached`.

The arguments to the action are...

- `handle` handle to the plug-in instance, cast to an *OfxImageEffectHandle*
- `inArgs` is redundant and set to NULL
- `outArgs` is redundant and set to NULL

A plug-in can return...

- *kOfxStatOK*, the action was trapped and all was well
- *kOfxStatReplyDefault*, the action was ignored, but all was well anyway
- *kOfxStatErrMemory*, in which case this may be called again after a memory purge
- *kOfxStatFailed*, something went wrong, but no error code appropriate, the plug-in should to post a message if possible and the host should not attempt to run the plug-in in OpenGL render mode.

Typedefs

```
typedef struct OfxImageEffectOpenGLRenderSuiteV1 OfxImageEffectOpenGLRenderSuiteV1
```

OFX suite that provides image to texture conversion for OpenGL processing.

group **CudaRender**

Version

CUDA rendering was added in version 1.5.

Defines

kOfxImageEffectPropCudaRenderSupported

Indicates whether a host or plug-in can support CUDA render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support CUDA render
 - “true” - the host or plug-in can support CUDA render

kOfxImageEffectPropCudaEnabled

Indicates that a plug-in SHOULD use CUDA render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropCudaRenderSupported`=“true” then the host MAY set this property to indicate that it is passing images as CUDA memory pointers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the `kOfxImagePropData` of each image of each clip is a CPU memory pointer.
 - 1 indicates that the `kOfxImagePropData` of each image of each clip is a CUDA memory pointer.

kOfxImageEffectPropCudaStreamSupported

Indicates whether a host or plug-in can support CUDA streams.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - in which case the host or plug-in does not support CUDA streams
 - “true” - which means a host or plug-in can support CUDA streams

kOfxImageEffectPropCudaStream

The CUDA stream to be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property will only be set if the host and plug-in both support CUDA streams.

If set:

- this property contains a pointer to the stream of CUDA render (`cudaStream_t`). In order to use it, `reinterpret_cast<cudaStream_t>(pointer)` is needed.
- the plug-in SHOULD ensure that its render action enqueues any asynchronous CUDA operations onto the supplied queue.
- the plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action, and SHOULD NOT call `cudaDeviceSynchronize()` at any time.

If not set:

- the plug-in SHOULD ensure that any asynchronous operations it enqueues have completed before returning from the render action.

group **MetalRender**

Version

Metal rendering was added in version 1.5.

Defines

kOfxImageEffectPropMetalRenderSupported

Indicates whether a host or plug-in can support Metal render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support Metal render
 - “true” - the host or plug-in can support Metal render

kOfxImageEffectPropMetalEnabled

Indicates that a plug-in SHOULD use Metal render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropMetalRenderSupported=“true”` then the host MAY set this property to indicate that it is passing images as Metal buffers.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*
- Valid Values
 - 0 indicates that the *kOfxImagePropData* of each image of each clip is a CPU memory pointer.
 - 1 indicates that the *kOfxImagePropData* of each image of each clip is a Metal id<MTLBuffer>.

kOfxImageEffectPropMetalCommandQueue

The command queue of Metal render.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - *kOfxImageEffectActionRender*
 - *kOfxImageEffectActionBeginSequenceRender*
 - *kOfxImageEffectActionEndSequenceRender*

This property contains a pointer to the command queue to be used for Metal rendering (id<MTLCommandQueue>). In order to use it, reinterpret_cast<id<MTLCommandQueue>>(pointer) is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous Metal operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

group **OpenCLRender**

Version

OpenCL rendering was added in version 1.5.

Defines

kOfxImageEffectPropOpenCLRenderSupported

Indicates whether a host or plug-in can support OpenCL Buffers render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - “false” for a plug-in
- Valid Values - This must be one of
 - “false” - the host or plug-in does not support OpenCL Buffers render

- "true" - the host or plug-in can support OpenCL Buffers render

kOfxImageEffectPropOpenCLSupported

Indicates whether a host or plug-in can support OpenCL Images render.

- Type - string X 1
- Property Set - plug-in descriptor (read/write), host descriptor (read only)
- Default - "false" for a plug-in
- Valid Values - This must be one of
 - "false" - in which case the host or plug-in does not support OpenCL Images render
 - "true" - which means a host or plug-in can support OpenCL Images render

kOfxImageEffectPropOpenCLEnabled

Indicates that a plug-in SHOULD use OpenCL render in the current action.

If a plug-in and host have both set `kOfxImageEffectPropOpenCLRenderSupported="true"` or have both set `kOfxImageEffectPropOpenCLSupported="true"` then the host MAY set this property to indicate that it is passing images as OpenCL Buffers or Images.

When rendering using OpenCL Buffers, the `cl_mem` of the buffers are retrieved using `kOfxImagePropData`. When rendering using OpenCL Images, the `cl_mem` of the images are retrieved using `kOfxImageEffectPropOpenCLImage`. If both `kOfxImageEffectPropOpenCLSupported` (Buffers) and `kOfxImageEffectPropOpenCLRenderSupported` (Images) are enabled by the plug-in, it should use `kOfxImageEffectPropOpenCLImage` to determine which is being used by the host.

- Type - int X 1
- Property Set - inArgs property set of the following actions...
 - `kOfxImageEffectActionRender`
 - `kOfxImageEffectActionBeginSequenceRender`
 - `kOfxImageEffectActionEndSequenceRender`
- Valid Values
 - 0 indicates that a plug-in SHOULD use OpenCL render in the render action
 - 1 indicates that a plug-in SHOULD NOT use OpenCL render in the render action

kOfxImageEffectPropOpenCLCommandQueue

Indicates the OpenCL command queue that should be used for rendering.

- Type - pointer X 1
- Property Set - inArgs property set of the following actions...
 - `kOfxImageEffectActionRender`
 - `kOfxImageEffectActionBeginSequenceRender`
 - `kOfxImageEffectActionEndSequenceRender`

This property contains a pointer to the command queue to be used for OpenCL rendering (`cl_command_queue`). In order to use it, `reinterpret_cast<cl_command_queue>(pointer)` is needed.

The plug-in SHOULD ensure that its render action enqueues any asynchronous OpenCL operations onto the supplied queue.

The plug-in SHOULD NOT wait for final asynchronous operations to complete before returning from the render action.

kOfxImageEffectPropOpenCLImage

Indicates the image handle of an image supplied as an OpenCL Image by the host.

- Type - pointer X 1
- Property Set - image handle returned by `clipGetImage`

This value should be cast to a `cl_mem` and used as the image handle when performing OpenCL Images operations. The property should be used (not *kOfxImagePropData*) when rendering with OpenCL Images (*kOfxImageEffectPropOpenCLSupported*), and should be used to determine whether Images or Buffers should be used if a plug-in supports both *kOfxImageEffectPropOpenCLSupported* and *kOfxImageEffectPropOpenCLRenderSupported*. Note: the `kOfxImagePropRowBytes` property is not required to be set by the host, since OpenCL Images do not have the concept of row bytes.

kOfxOpenCLProgramSuite

Typedefs

typedef struct *OfxOpenCLProgramSuiteV1* **OfxOpenCLProgramSuiteV1**

OFX suite that allows a plug-in to get OpenCL programs compiled.

This is an optional suite the host can provide for building OpenCL programs for the plug-in, as an alternative to calling `clCreateProgramWithSource / clBuildProgram`. There are two advantages to doing this: The host can add flags (such as `-cl-denorms-are-zero`) to the build call, and may also cache program binaries for performance (however, if the source of the program or the OpenCL environment changes, the host must recompile so some mechanism such as hashing must be used).

group **ImageEffectActions**

These are the list of actions passed to an image effect plugin's main function. For more details on how to deal with actions, see Image Effect Actions.

group **ImageEffectPropDefines**

These are the list of properties used by the Image Effects API.

group **StatusCodesImageEffect**

These are status codes returned by functions in the `OfxImageEffectSuite` and Image Effect plugin functions.

They range from 1000 until 1999

group **PropertiesInteract**

These are the list of properties used by the Interact API documented in `CustomInteractionPage`.

group InteractActions

These are the list of actions passed to an interact's entry point function. For more details on how to deal with actions, see Interact Actions.

group KeySyms

These keysymbols are used as values by the *kOfxPropKeySym* property to indicate the value of a key that has been pressed. A corresponding *kOfxPropKeyString* property is also set to contain the unicode value of the key (if it has one).

The special keysym *kOfxKey_Unknown* is used to set the *kOfxPropKeySym* property in cases where the key has a UTF8 value which is not supported by the symbols below.

group ParamTypeDefines

These strings are used to identify the type of the parameter when it is defined, they are also on the *kOfxParamPropType* in any parameter instance.

group ParamPropDefines

These are the list of properties used by the parameters suite.

group ErrorCodes**page ofxOpenGLRender****1.22.1 Introduction**

The *OfxOpenGLRenderSuite* allows image effects to use OpenGL commands (hopefully backed by a GPU) to accelerate rendering of their outputs. The basic scheme is simple...

- An effect indicates it wants to use OpenGL acceleration by setting the *kOfxImageEffectPropOpenGLRenderSupported* flag on its descriptor
- A host indicates it supports OpenGL acceleration by setting *kOfxImageEffectPropOpenGLRenderSupported* on its descriptor
- In an effect's *kOfxImageEffectActionGetClipPreferences* action, an effect indicates what clips it will be loading images from onto the GPU's memory during an effect's *kOfxImageEffectActionRender* action.

1.22.2 OpenGL House Keeping

If a host supports OpenGL rendering then it flags this with the string property *kOfxImageEffectPropOpenGLRenderSupported* on its descriptor property set. Effects that cannot run without OpenGL support should examine this in *kOfxActionDescribe* action and return a *kOfxStatErrMissingHostFeature* status flag if it is not set to "true".

Effects flag to a host that they support OpenGL rendering by setting the string property *kOfxImageEffectPropOpenGLRenderSupported* on their effect descriptor during the *kOfxActionDescribe* action. Effects can work in three ways...

- purely on CPUs without any OpenGL support at all, in which case they should set *kOfxImageEffectPropOpenGLRenderSupported* to be "false" (the default),
- on CPUs but with optional OpenGL support, in which case they should set *kOfxImageEffectPropOpenGLRenderSupported* to be "true",

- only with OpenGL support, in which case they should set *kOfxImageEffectPropOpenGLRenderSupported* to be “needed”.

Hosts can examine this flag and respond to it appropriately.

Effects can use OpenGL accelerated rendering during the following action...

- *kOfxImageEffectActionRender*

If an effect has indicated that it optionally supports OpenGL acceleration, it should check the property *kOfxImageEffectPropOpenGLEnabled* passed as an in argument to the following actions,

- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*

If this property is set to 0, then it should not attempt to use any calls to the OpenGL suite or OpenGL calls whilst rendering.

1.22.3 Getting Images as Textures

An effect could fetch an image into memory from a host via the standard Image Effect suite “clipGetImage” call, then create an OpenGL texture from that. However as several buffer copies and various other bits of house keeping may need to happen to do this, it is more efficient for a host to create the texture directly.

The *OfxOpenGLRenderSuiteV1::clipLoadTexture* function does this. The arguments and semantics are similar to the *OfxImageEffectSuiteV2::clipGetImage* function, with a few minor changes.

The effect is passed back a property handle describing the texture. Once the texture is finished with, this should be disposed of via the *OfxOpenGLRenderSuiteV1::clipFreeTexture* function, which will also delete the associated OpenGL texture (for source clips).

The returned handle has a set of properties on it, analogous to the properties returned on the image handle by *OfxImageEffectSuiteV2::clipGetImage*. These are:

- *kOfxImageEffectPropOpenGLTextureIndex*
- *kOfxImageEffectPropOpenGLTextureTarget*
- *kOfxImageEffectPropPixelDepth*
- *kOfxImageEffectPropComponents*
- *kOfxImageEffectPropPreMultiplication*
- *kOfxImageEffectPropRenderScale*
- *kOfxImagePropPixelAspectRatio*
- *kOfxImagePropBounds*
- *kOfxImagePropRegionOfDefinition*
- *kOfxImagePropRowBytes*
- *kOfxImagePropField*
- *kOfxImagePropUniqueIdentifier*

The main difference between this and an image handle is that the *kOfxImagePropData* property is replaced by the *kOfxImageEffectPropOpenGLTextureIndex* property. This integer property should be cast to a *GLuint* and is the index to use for the OpenGL texture. Next to texture handle the texture target enumerator is given in *kOfxImageEffectPropOpenGLTextureTarget*

Note, because the image is being directly loaded into a texture by the host it need not obey the Clip Preferences action to remap the image to the pixel depth the effect requested.

1.22.4 Render Output Directly with OpenGL

Effects can use the graphics context as they see fit. They may be doing several render passes with fetch back from the card to main memory via ‘render to texture’ mechanisms interleaved with passes performed on the CPU. The effect must leave output on the graphics card in the provided output image texture buffer.

The host will create a default OpenGL viewport that is the size of the render window passed to the render action. The following code snippet shows how the viewport should be rooted at the bottom left of the output texture.

```
// set up the OpenGL context for the render to texture
...

// figure the size of the render window
int dx = renderWindow.x2 - renderWindow.x1;
int dy = renderWindow.y2 - renderWindow.y2;

// setup the output viewport
glViewport(0, 0, dx, dy);
```

Prior to calling the render action the host may also choose to bind the output texture as the current color buffer (render target), or they may defer doing this until clipLoadTexture is called for the output clip.

After this, it is completely up to the effect to choose what OpenGL operations to render with, including projections and so on.

1.22.5 OpenGL Current Context

The host is only required to make the OpenGL context current (e.g., using wglMakeCurrent, for Windows) during the following actions:

- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*
- *kOfxActionOpenGLContextAttached*
- *kOfxActionOpenGLContextDetached*

For the first 3 actions, Render through EndSequenceRender, the host is only required to set the OpenGL context if *kOfxImageEffectPropOpenGLEnabled* is set. In other words, a plug-in should not expect the OpenGL context to be current for other OFX calls, such as *kOfxImageEffectActionDescribeInContext*.

page **ofxOpenCLRender**

1.22.6 Introduction

The OpenCL extension enables plug-ins to use OpenCL commands (typically backed by a GPU) to accelerate rendering of their outputs. The basic scheme is simple. . . .

- an plug-in indicates it wants to use OpenCL acceleration by setting the *kOfxImageEffectPropOpenCLSupported* (Images) and/or *kOfxImageEffectPropOpenCLRenderSupported* (Buffers) flags on it's descriptor.
- a host indicates it supports OpenCL acceleration by setting *kOfxImageEffectPropOpenCLSupported* (Images) and/or *kOfxImageEffectPropOpenCLRenderSupported* (Buffers) on it's descriptor.
- the host decides when to use OpenCL, and sets the *kOfxImageEffectPropOpenCLEnabled* property on the *BeginRender/Render/EndRender* calls to indicate this.
- when OpenCL Images are being used (*kOfxImageEffectPropOpenCLSupported*) the clip image property *kOfxImageEffectPropOpenCLImage* will be set and non-null.
- when OpenCL Buffers are being used (*kOfxImageEffectPropOpenCLRenderSupported*) the clip image property *kOfxImagePropData* will be set and non-null.

1.22.7 Discovery and Enabling

If a host supports OpenCL rendering then it flags with the string property *kOfxImageEffectPropOpenCLSupported* (Images) and/or *kOfxImageEffectPropOpenCLRenderSupported* (Buffers) on its descriptor property set. Effects that cannot run without OpenCL support should examine this in *kOfxActionDescribe* action and return a *kOfxStatErrMissingHostFeature* status flag if it is not set to "true".

Effects flag to a host that they support OpenCL rendering by setting the string property *kOfxImageEffectPropOpenCLSupported* (Images) and/or *kOfxImageEffectPropOpenCLRenderSupported* (Buffers) on their effect descriptor during the *kOfxActionDescribe* action. Effects can work in two ways. . . .

- purely on CPUs without any OpenCL support at all, in which case they should set *kOfxImageEffectPropOpenCLSupported* (Images) and *kOfxImageEffectPropOpenCLRenderSupported* (Buffers) to be "false" (the default),
- on CPUs but with optional OpenCL support, in which case they should set *kOfxImageEffectPropOpenCLSupported* (Images) and/or *kOfxImageEffectPropOpenCLRenderSupported* (Buffers) to be "true"

Host may support just OpenCL Images, just OpenCL Buffers, or both, as indicated by which of these two properties they set "true". Likewise plug-ins may support just OpenCL Images, just OpenCL Buffers, or both, as indicated by which of these two properties they set "true". If both host and plug-in support both, it is up to the host which it uses. Typically, it will be based on what it uses natively (to avoid an extra copy operation). If a plug-in supports both, it must use *kOfxImageEffectPropOpenCLImage* to determine if Images or Buffers are being used for a given render action.

Effects can use OpenCL render only during the following action:

- *kOfxImageEffectActionRender*

If a plug-in has indicated that it optionally supports OpenCL acceleration, it should check the property *kOfxImageEffectPropOpenCLEnabled* passed as an in argument to the following actions,

- *kOfxImageEffectActionRender*
- *kOfxImageEffectActionBeginSequenceRender*
- *kOfxImageEffectActionEndSequenceRender*

If this property is set to 0, then it must not attempt to use OpenCL while rendering. If this property is set to 1, then it must use OpenCL buffers or images while rendering.

If a call using OpenCL rendering fails, the host may re-attempt using CPU buffers instead, but this is not required, and might not be efficient.

1.22.8 OpenCL platform and device versions and feature support

Assume an in-order command queue. Do not assume a profiling command queue.

Effects should target OpenCL 1.1 API and OpenCL C kernel language support. Only minimum required features required in OpenCL 1.1 should be used (for example, see “5.3.2.1 Minimum List of Supported Image Formats” for the list of image types which can be expected to be supported across all devices). If you have specific requirements for features beyond these minimums, you will need to check the device (e.g., using `clGetDeviceInfo` with `CL_DEVICE_EXTENSIONS`) to see if your feature is available, and have a fallback if it's not.

Temporary buffers and images should not be kept past the render action. A separate extension for host managed caching is in the works.

Do not retain OpenCL objects without a matching release within the render action.

1.22.9 Multiple OpenCL Devices

This is very important: The host may support multiple OpenCL devices. Therefore the plug-in should keep a separate set of kernels per OpenCL content (e.g., using a map). The OpenCL context can be found from the command queue using `clGetCommandQueueInfo` with `CL_QUEUE_CONTEXT`. Failure to do this will cause crashes or incorrect results when the host switches to another OpenCL device.

page deprecated

Member *kOfxImageComponentYUVA*

- removed in v1.4. Note, this has been deprecated in v1.3

Member *kOfxImageEffectPropInAnalysis*

- This feature has been deprecated - officially commented out v1.4.

Member *kOfxInteractPropViewportSize*

- V1.3: This property is the redundant and its use will be deprecated in future releases. V1.4: Removed

Member *kOfxParamDoubleTypeNormalisedX*

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeX` V1.4: Removed

Member *kOfxParamDoubleTypeNormalisedXAbsolute*

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeXAbsolute` V1.4: Removed

Member *kOfxParamDoubleTypeNormalisedXY*

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeXY` V1.4: Removed

Member *kOfxParamDoubleTypeNormalisedXYAbsolute*

- V1.3: Deprecated in favour of *kOfxParamDoubleTypeXYAbsolute* V1.4: Removed

Member *kOfxParamDoubleTypeNormalisedY*

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeY` V1.4: Removed

Member *kOfxParamDoubleTypeNormalisedYAbsolute*

- V1.3: Deprecated in favour of `::OfxParamDoubleTypeYAbsolute` V1.4: Removed

Member *kOfxParamPropPluginMayWrite*

- v1.4: deprecated - to be removed in 1.5

Member *OfxYUVAColourF*

- Deprecated in 1.3, removed in 1.4

Member *OfxYUVAColourS*

- Deprecated in 1.3, removed in 1.4

page **index**

This page represents the automatically extracted HTML documentation of the source headers for the OFX Image Effect API. The documentation was extracted by doxygen (<http://www.doxygen.org>). A more complete reference manual is <https://openfx.readthedocs.io>.

1.23 Status Codes

Status codes are returned by most functions in OFX suites and all plug-in actions to indicate the success or failure of the operation. All status codes are defined in *ofxCore.h* and *#defined* to be integers.

typedef int **OfxStatus**

OFX status return type.

Most OFX functions in host suites and all actions in a plug-in return a status code, where the status codes are all 32 bit integers. This typedef is used to label that status code.

kOfxStatOK

Status code indicating all was fine.

kOfxStatFailed

Status error code for a failed operation.

kOfxStatErrFatal

Status error code for a fatal error.

Only returned in the case where the plug-in or host cannot continue to function and needs to be restarted.

kOfxStatErrUnknown

Status error code for an operation on or request for an unknown object.

kOfxStatErrMissingHostFeature

Status error code returned by plug-ins when they are missing host functionality, either an API or some optional functionality (eg: custom params).

Plug-Ins returning this should post an appropriate error message stating what they are missing.

kOfxStatErrUnsupported

Status error code for an unsupported feature/operation.

kOfxStatErrExists

Status error code for an operation attempting to create something that exists.

kOfxStatErrFormat

Status error code for an incorrect format.

kOfxStatErrMemory

Status error code indicating that something failed due to memory shortage.

kOfxStatErrBadHandle

Status error code for an operation on a bad handle.

kOfxStatErrBadIndex

Status error code indicating that a given index was invalid or unavailable.

kOfxStatErrValue

Status error code indicating that something failed due an illegal value.

kOfxStatReplyYes

OfxStatus returned indicating a 'yes'.

kOfxStatReplyNo

OfxStatus returned indicating a 'no'.

kOfxStatReplyDefault

OfxStatus returned indicating that a default action should be performed.

1.23.1 Status codes for GPU renders:

These are defined in `ofxGPURender.h`.

kOfxStatGPUOutOfMemory

GPU render ran out of memory.

kOfxStatGLOutOfMemory

OpenGL render ran out of memory (same as `kOfxStatGPUOutOfMemory`)

kOfxStatGPURenderFailed

GPU render failed in a non-memory-related way.

kOfxStatGLRenderFailed

OpenGL render failed in a non-memory-related way (same as `kOfxStatGPURenderFailed`)

1.24 Changes to the API for 1.2

1.24.1 Introduction

This chapter lists the changes and extensions between the 1.1 version of the API and the 1.2 version of the API. The extension are backwards compatible, so that a 1.2 plugin will run on an earlier version of a host, provided a bit of care is taken. A 1.2 host will easily support a plugin written to an earlier API.

1.24.2 Packaging

A new architecture directory was added to the bundle hierarchy to specifically contain Mac OSX 64 bit builds. The current 'MacOS' architecture is a fall back for 32 bit only and/or universal binary builds.

1.24.3 Versioning

Three new properties are provided to identify and version a plugin and/or host. These are...

- *kOfxPropAPIVersion* a multi-dimensional integer that specifies the version of the API being implemented by a host.
- *kOfxPropVersion* a multi-dimensional integer that provides a version number for host and plugin
- *kOfxPropVersionLabel* a user readable version label

Before 1.2 there was no way to identify the version of a host application, which a plugin could use to work around known bugs and problems in known versions. *kOfxPropVersion* provides a way to do that.

1.24.4 Plugin Description

The new property *kOfxPropPluginDescription* allows a plugin to set a string which provides a description to a user.

1.24.5 Parameter Groups and Icons

Group parameters are typically displayed in a hierarchical manner on many applications, with 'twirlies' to open and close the group. The new property *kOfxParamPropGroupOpen* is used to specify if a group parameter should be initially open or closed.

Some applications are able to display icons instead of text when labelling parameters. The new property, *kOfxPropIcon*, specifies an SVG file and/or a PNG file to use as an icon in such host applications.

1.24.6 New Message Suite

A new message suite has been specified, *OfxMessageSuiteV2*, this adds two new functions. One to set a persistent message on an effect, and a second to clear that message. This would typically be used to flag an error on an effects.

1.24.7 New Syncing Property

A new property has been added to parameter sets, *kOfxPropParamSetNeedsSyncing*. This is used by plugins with internal data structures that need syncing back to parameters for persistence and so on. This property should be set whenever the plugin changes it's internal state to inform the host that a sync will be required before the next serialisation of the plugin. Without this property, the host would have to continually force the plugin to sync it's private data, whether that was a redundant operation or not. For large data sets, this can be a significant overhead.

1.24.8 Sequential Rendering

Flagging sequential rendering has been slightly modified. The *kOfxImageEffectInstancePropSequentialRender* property has had a third allowed state added, which indicate that a plugin would prefer to be sequentially rendered if possible, but need not be.

The *kOfxImageEffectInstancePropSequentialRender* property has also been added to the host descriptor, to indicate whether the host can support sequential rendering.

The new property *kOfxImageEffectPropSequentialRenderStatus* is now passed to the render actions to indicate that a host is currently sequentially rendering or not.

1.24.9 Interactive Render Notification

A new property has been added to flag a render as being in response to an interactive change by a user, as opposed to a batch render. This is *kOfxImageEffectPropInteractiveRenderStatus*

1.24.10 Host Operating System Handle

A new property has been added to allow a plugin to request the host operating system specific application handle (ie: on Windows (tm) this would be the application's root hWnd). This is *kOfxPropHostOSHandle*

1.24.11 Non Normalised Spatial Parameters

Normalised double parameters have proved to be more of a problem than expected. The major idea was to provide resolution independence for spatial parameters. However, in practice, having to specify parameters as a fraction of a yet to be determined resolution is problematic. For example, if you want to set something to be explicitly '20', there is no way of doing that. The main problem stems from normalised params conflating two separate issues, flagging to the host that a parameter was spatial, and being able to specify defaults in a normalised co-ordinate system.

With 1.2 new spatial double parameter types are defined. These have their values manipulated in canonical coordinates, however, they have an option to specify their default values in a normalise coordinate system. These are...

These new double parameter types are...

- *kOfxParamDoubleTypeX* - a size in the X dimension dimension (1D only), new for 1.2
- *kOfxParamDoubleTypeXAbsolute* - a position in the X dimension (1D only), new for 1.2
- *kOfxParamDoubleTypeY* - a size in the Y dimension dimension (1D only), new for 1.2
- *kOfxParamDoubleTypeYAbsolute* - a position in the X dimension (1D only), new for 1.2
- *kOfxParamDoubleTypeXY* - a size in the X and Y dimension (2D only), new for 1.2
- *kOfxParamDoubleTypeXYAbsolute* - a position in the X and Y dimension (2D only), new for 1.2

These new parameter types can set their defaults in one of two coordinate systems, the property *kOfxParamPropDefaultCoordinateSystem* Specifies the coordinate system the default value is being specified in.

Plugins can check *kOfxPropAPIVersion* to see if these new parameter types are supported

1.24.12 Native Overlay Handles

Some applications have their own overlay handles for certain types of parameter (eg: spatial positions). It is often better to rely on those, than have a plugin implement their own overlay handles. Two new parameter, properties are available to do that, one used by the host to indicate if such handles are available. The other by a plugin telling the host to use such handle.

- *kOfxParamPropHasHostOverlayHandle* indicates a parameter has an host native overlay handle
- *kOfxParamPropUseHostOverlayHandle* indicates that a host should use a native overlay handle.

1.24.13 Interact Colour Hint

Some applications allow the user to specify colours of any overlay via a colour picker. Plug-ins can access this via the *kOfxInteractPropSuggestedColour* property.

1.24.14 Interact Viewport Pen Position

The new property *kOfxInteractPropPenViewportPosition* is used to pass a pen position in viewport coordinate, rather than a connaonical. This is sometimes much more convenient. It is passed to all actions that *kOfxInteractPropPenPosition* is passed to.

1.24.15 Parametric Parameters

A new optional parameter type, and supporting suite, is introduced, parametric parameters. This allows for the construction of user defined lookup tables and so on.

OPENFX PROGRAMMING GUIDE

This is a mostly complete reference guide to the OFX image effect plugin architecture. It is a backwards compatible update to the 1.2 version of the API.

2.1 Foreword

OFX is an open API for writing visual effects plug-ins for a wide variety of applications, such as video editing systems and compositing systems. This guide demonstrates by example the low level C APIs that defines OFX.

2.2 Intended Audience

Do you write visual effects or image processing software? Do you have an application which deals with moving images and hosts plug-ins or would like to host plug-ins? Then OFX is for you.

This guide assumes you can program in the C language and are familiar with the concepts involved in writing visual effects software. You need to understand concepts like pixels, clips, pixel aspect ratios and so on. If you don't, I suggest you read further or attempt to soldier on bravely and see how far you go before you get lost.

2.3 What is OFX?

OFX is actually several things. At the lowest level OFX is a generic C based plug-in architecture that can be used to define any kind of plug-in API. You could use this low level architecture to implement any API, however it was originally designed to host our visual effects image processing API. The basic architecture could be re-used to create other higher level APIs such as a sound effects API, a 3D API and more.

This guide describes the basic OFX plug-in architecture and the visual effects plug-in API built on top of it. The visual effects API is very broad and intended to allow visual effects plug-ins to work on a wide range of host applications, including compositing hosts, rotoscopers, encoding applications, colour grading hosts, editing hosts and more I haven't thought of yet. While all these type of applications process images, they often have very different work flows and present effects to a user in incompatible ways. OFX is an attempt to deal with all of these in a clear and consistent manner.

The API is by design feature rich, not all aspects of the API map to all hosts. This is to allow different host developers to implement OFX support in a manner that best fits their applications' capabilities.

Hosts are encouraged to extend OFX by providing extra proprietary suites, actions, properties and settings to extend the capabilities of the API. It would be nice that a broadly useful proprietary extension be put forward for incorporation into the open standard.

That said although there is no validation process in terms of what is an OFX host, a small minimal set of expectations is assumed, which we will cover in the following guides.

2.4 The Examples

I'll illustrate the API and how it works with a variety of example plugins, each of which will have its own guide describing what is going on. You should work through them one by one as each will build on the one before.

For completeness and clarity of explanation, each plugin is entirely self contained and has no dependency on anything other than standard C and C++ libraries and the OFX headers.

- *The basic machinery of an OFX plugin.*
- *How to access images.*
- *How to define parameters.*
- *How to write multi context effects.*
- *Coordinate systems and defining regions of definition.*

2.5 Wrapping the API

This API can be somewhat awkward to use directly, and it is expected that most plugin or host developers will wrap the API in higher level C or C++ structures.

There are open source host and plugin side API C wrappers available from the [openfx.org git repository](https://openfx.org). As you work through the examples you'll see that I actually start wrapping up various entities within the API into C classes as it can get unwieldy otherwise.

2.6 License

Please feel free to use any of the code you find here, provided you adhere to the BSD style license you'll find at the top of each header file.

This is a guide to the basic machinery an OFX plugin uses to communicate with a host application, and goes into the fundamentals of the API.

An example plugin will be used to illustrate how all the machinery works, and its source can be found in the C++ file [there](#). This plugin is a *no-op* image effect and does absolutely nothing to images, it is there purely to show you the basics of how a host and plugin work together. I'll embed snippets of the plugin, but with some comments and debug code stripped for clarity.

An OFX plugin is a compiled dynamic library that an application can load on demand to add extra features to itself. A standardised API is used by a host and a plugin to communicate and do what is needed.

OFX has an underlying plugin mechanism that could be used to create a wide variety of plugin APIs, but currently only one has been layered on top of the base plugin machinery, which is the OFX Image Effect API.

The OFX API is specified using the C programming language purely by a set of header files, there are no libraries a plugin need to link against to make a plugin or host work.¹

¹ Though there exist optional host and plugin support libraries that can be used to help you in your coding.

2.7 Key Concepts and Terminology

OFX has several key concepts and quite specific terminology, which definitely need defining.

- a **host** is an application than can load OFX plugins and provides an environment for plugins to work in,
- a **plugin** provides a set of extra features to a host application,
- a **binary** is a dynamic library² that contains one or more plugins,
- a **suite** is C `struct` containing a set of function pointers, which are named and versioned, Suites are the way a host allows a plugin to call functions within it and not have to link against anything,
- a **property** is a named object of a restricted set of C types, which is accessed via a property suite,
- a **property set** is a collection of properties,
- an **action** is a call into a plugin to do something,
- an **API** is a collection of suites, actions and properties that are used to do something useful, like process images. APIs are named and versioned.

2.8 The Two Bootstrapper Functions

To tell the host what it has inside it, a plugin binary needs to expose two functions to bootstrap the whole host/plugin communications process. These are:

- *OfxGetNumberOfPlugins()* A function that returns the number of plugins within that binary
- *OfxGetPlugin()* A function that returns a `struct` that provides the information required by a host to bootstrap the plugin.

The host should load the binary using the appropriate operating system calls, then search it for these two exposed symbols. It should then iterate over the number of advertised plugins and decide what to do with the plugins it finds.

It should go without saying that a host should not hang onto the pointer returned by *OfxGetPlugin()* after it unloads a binary, as the data will not be valid. It should also copy any strings out of the struct if it wants to keep them.

From our example, we have the following...

basics.cpp

```
// how many plugins do we have in this binary?
int OfxGetNumberOfPlugins(void)
{
    return 1;
}

// return the OfxPlugin struct for the nth plugin
OfxPlugin * OfxGetPlugin(int nth)
{
    if(nth == 0)
        return &effectPluginStruct;
    return 0;
}
```

² which will be operating system specific

2.9 The OfxPluginStruct

The *OfxPlugin* returned by *OfxGetPlugin()* provides information about the implementation of a particular plugin.

The fields in the struct give the host enough information to uniquely identify the plugin, what it does, and what version it is. These are:

- **pluginAPI** - the name of the API that this plugin satisfies, image effect plugins should set this to *kOfxImageEffectPluginApi*,
- **apiVersion** - the version of that API the plug-in was written to
- **pluginIdentifier** - the unique name of the plug-in. Used only to disambiguate the plug-in from all other plug-ins, not necessarily for human eyes
- **pluginVersionMajor** - the major version of the plug-in, typically incremented when compatibility breaks,
- **pluginVersionMinor** - the minor version of the plug-in, typically incremented when bugs and so on are fixed,
- **setHost** - a function used to set the *OfxHost struct* in the plugin,
- **mainEntry** - the function a host will use to send action requests to the plugin.

Our example plugin's *OfxPlugin* struct looks like...

basics.cpp

```
static OfxPlugin effectPluginStruct =
{
    kOfxImageEffectPluginApi,
    1,
    "org.openeffects:BasicsExamplePlugin",
    1,
    0,
    SetHostFunc,
    MainEntryPoint
};
```

Using this information a host application can grab a plugin struct then figure out if it supports the API at the given version.

The **pluginIdentifier** is not meant to be presented to the user, it is a purely a unique id for that plugin, *and any related versions* of that plugin. Use this for serialisation etc... to identify the plugin. The domainname:pluginname nomenclature is suggested best practice for a unique id. For a user visible name, use the *kOfxPropVersionLabel* property

Plugin versioning allows a plugin (as identified by the **pluginIdentifier** field) to be updated and redistributed multiple times, with the host knowing which is the most appropriate version to use. It even allows old and new versions of the same plugin to be used simultaneously within a host application. There are more details on how to use the version numbers in the OFX Programming Reference.

The **setHost** function is used by the host to give the plugin an *OfxHost* struct (see below), which is the bit that gives the plugin access to functions within the host application.

Finally the *mainEntry* is the function called by the host to get the plugin to carry out actions. Via the property system it behaves as a generic function call, allowing arbitrary numbers of parameters to be passed to the plugin.

2.10 Suites

A suite is simply a struct with a set of function pointers. Each suite is defined by a C struct definition in an OFX header file, as well as a C literal string that names the suite. A host will pass a set of suites to a plugin, each suite having the set of function pointers filled appropriately.

For example, look in the file `ofxMemory.h` for the suite used to perform memory allocation:

`ofxMemory.h`

Notice also, the version number built into the name of the memory suite. If we ever needed to change the memory suite for some reason, `OfxMemorySuiteV2` would be defined, with a new set of function pointers. The new suite could then live along side the old suite to provide backwards compatibility.

Plugins have to ask for suites from the host by name with a specific version, how we do that is covered next.

2.10.1 The OfxHost and Fetching Suites

An instance of an `OfxHost` C struct is the thing that allows a plugin to get suites and provides information about a host application

A plugin is given one of these by the host application via the `OfxPlugin::setHost()` function it previously passed to the host.

There are two members to an `OfxHost`, the first is a property set (more on properties in a moment) which describes what the host does and how it behaves.

The second member is a function used to fetch suites from the host application. Going back to our example plugin, we have the following bits of code. For the moment ignore how and when the `LoadAction` is called, but notice what it does...

`basics.cpp`

```
// The anonymous namespace is used to hide symbols from export.
namespace {
    OfxHost          *gHost;
    OfxPropertySuiteV1 *gPropertySuite = 0;
    OfxImageEffectSuiteV1 *gImageEffectSuite = 0;

    ///////////////////////////////////////////////////////////////////
    /// call back passed to the host in the OfxPlugin struct to set our host pointer
    void SetHostFunc(OfxHost *hostStruct)
    {
        gHost = hostStruct;
    }

    ///////////////////////////////////////////////////////////////////
    /// the first action called
    OfxStatus LoadAction(void)
    {
        gPropertySuite = (OfxPropertySuiteV1 *) gHost->fetchSuite(gHost->host,
                                                                    kOfxPropertySuite,
                                                                    1);
        gImageEffectSuite = (OfxImageEffectSuiteV1 *) gHost->fetchSuite(gHost->host,
                                                                           kOfxImageEffectSuite,
                                                                           1);
    }
}
```

(continues on next page)

(continued from previous page)

```

    return kOfxStatOK;
}
}

```

Notice that it is fetching two suites by name from the host. Firstly the all important *kOfxPropertySuite* and then the *kOfxImageEffectSuite*. It squirrels these away for later use in two global pointers. The plugin can then use the functions in the suites as and when needed.

2.11 Properties

The main way plugins and hosts communicate is via the properties mechanism. A property is a named object inside a property set, which is a bit like a python dictionary. You use the property suite, defined in the header **ofxProperty.h** to access them.

Properties can be of the following fundamental types...

- int
- double
- char *
- void *

So for in our example we have...

basics.cpp

```

OfxPropertySetHandle effectProps;
gImageEffectSuite->getPropertySet(effect, &effectProps);

gPropertySuite->propSetString(effectProps, kOfxPropLabel, 0, "OFX Basics Example");

```

Here the plugin is using the effect suite to get the property set on the effect. It is then setting the string property **kOfxPropLabel** to be “OFX Basics Example”. There are corresponding calls for the other data types, and equivalent set calls. All pretty straight forwards.

Notice the **0** passed as the third argument, which is an index. Properties can be multidimensional, for example the current pen position in a graphics viewport is a 2D integer property. You can get and set individual elements in a multidimensional property or you could use calls like **OfxPropertySuiteV1::propSetIntN** to set all values at once. Of course there exists *N* calls for all types, as well as corresponding setting calls.

The various OFX header files are littered with C macros that define the properties used by the API, what type they are, what property set they are on and whether you can read and/or write them. The OFX reference guide had all the properties listed by name and object they are on, as well as what they are for.

By passing information via property sets, rather than fixed C structs, you gain a flexibility that allows for simple incremental additions to the API without breaking backwards compatibility and builds. It does come at a cost (being continual string look-up), but the flexibility it gives is worth it.

Note: Plugins have to be very careful with scope of the pointer returned when you fetch a string property. The pointer will be guaranteed to be valid *only* until the next call to an OFX suite function or until the action ends. If you want to use the string out of those scope you *must* copy it.

2.12 Actions

Actions are how a host tells a plugin what to do. The *mainEntry* function pointer in the *OfxPlugin* structure is the what accepts actions to do whatever is being requested.

Where:

- *action* is a C string that specifies what is to be done by the plugin, e.g. *OfxImageEffectActionRender* tells an image effect plugin to render a frame
- *handle* is the thing that is being operated on, and needs to be downcast appropriately, what this is will depend on the action
- *inArgs* is a well defined property set that are the arguments to the action
- *outArgs* is a well defined property set where a plugin can return values as needed.

The entry point will return an *OfxStatus* to tell the host what happened. A plugin is not obliged to trap all actions, just a certain subset, and if it doesn't need to trap the action, it can just return the status *kOfxStatReplyDefault* to have the host carry out the well defined default for that action.

So looking at our example we can see its main entry point:

basics.cpp

```
OfxStatus MainEntryPoint(const char *action,
                        const void *handle,
                        OfxPropertySetHandle inArgs,
                        OfxPropertySetHandle outArgs)
{
    // cast to appropriate type
    OfxImageEffectHandle effect = (OfxImageEffectHandle) handle;

    OfxStatus returnStatus = kOfxStatReplyDefault;

    if(strcmp(action, kOfxActionLoad) == 0) {
        returnStatus = LoadAction();
    }
    else if(strcmp(action, kOfxActionUnload) == 0) {
        returnStatus = UnloadAction();
    }
    else if(strcmp(action, kOfxActionDescribe) == 0) {
        returnStatus = DescribeAction(effect);
    }
    else if(strcmp(action, kOfxImageEffectActionDescribeInContext) == 0) {
        returnStatus = DescribeInContextAction(effect, inArgs);
    }
    else if(strcmp(action, kOfxActionCreateInstance) == 0) {
        returnStatus = CreateInstanceAction(effect);
    }
    else if(strcmp(action, kOfxActionDestroyInstance) == 0) {
        returnStatus = DestroyInstanceAction(effect);
    }
    else if(strcmp(action, kOfxImageEffectActionIsIdentity) == 0) {
        returnStatus = IsIdentityAction(effect, inArgs, outArgs);
    }
}
```

(continues on next page)

```

return returnStatus;
}

```

You can see the plugin is trapping seven actions and is saying to do the default for the rest of the actions.

In fact only four actions need to be trapped for an image effect plugin³, but our machinery plugin is trapping more for illustrative purposes.

What is on the property sets, and what the handle is depends on the action being called. Some actions have no arguments (eg: the `kOfxLoadAction`), while others have in and out arguments, e.g. the `kOfxImageEffectActionIsIdentity`.

Actions give us a very flexible and expandable generic function calling mechanism. This means it is trivial to expand the API via adding extra properties or actions to the API without impacting existing plugins or applications.

Note: For the main entry point on image effect plugins, the handle passed in will either be NULL or an `OfxImageEffectHandle`, which is just a blind pointer to host specific data that represents the plugin.

2.13 Basic Actions For Image Effect Plugins

There are a set of actions called on a plugin that signal to the plugin what is going on and to get it to tell the host what the plugin does. These need to be called in a specific sequence to make it all work properly.

2.13.1 The Load and Unload Actions

The `kOfxActionLoad` is the very first action passed to a plugin. It will be called after the `setHost` callback has been used to pass the `OfxHost` to the plugin. It is the point at which a plugin gets to create global structures that it will later be used across all instances. From our *load action snippet* above, you can see that the plugin is fetching two suites and caching the pointers away for later use.

At some point the host application will want to unload the binary that the plugin is contained in, either when the host quits or the plugin is no longer needed by the host application. The host needs to notify the plugin of this, as it may need to perform some clean up. The `kOfxActionUnload` action is sent to the plugin by the host to warn the plugin of its imminent demise. After this action is called the host can no longer issue any actions to that plugin unless another `kOfxActionLoad` action is called. In our example plugin, the unload does nothing.

Note: Hosts should always pair the `kOfxActionLoad` with a `kOfxActionUnload`, otherwise all sorts of badness can happen, including memory leaks, failing license checks and more. There is one exception to this, which is if a plugin encounters an error during the load action and returns an error state. In this case only, the plugin *must* clean up before it returns, and, the balancing unload action is *not* called. In all other circumstances where an error is returned by a plugin from any other action, the unload action will eventually be called.

³ `kOfxLoadAction`, `kOfxActionDescribe`, `kOfxImageEffectActionDescribeInContext` and one of `kOfxImageEffectActionIsIdentity` or `kOfxImageEffectActionRender`

2.13.2 Describing Plugins To A Host

Once a plugin has had *kOfxActionLoad* called on it, it will be asked to describe itself. This is done with the *kOfxActionDescribe* action. From our example plugin, here is the function called by our main entry point in response to the describe action.

basics.cpp

```
OfxStatus DescribeAction(OfxImageEffectHandle descriptor)
{
    // get the property set handle for the plugin
    OfxPropertySetHandle effectProps;
    gImageEffectSuite->getPropertySet(descriptor, &effectProps);

    // set some labels and the group it belongs to
    gPropertySuite->propSetString(effectProps,
                                  kOfxPropLabel,
                                  0,
                                  "OFX Basics Example");
    gPropertySuite->propSetString(effectProps,
                                  kOfxImageEffectPluginPropGrouping,
                                  0,
                                  "OFX Example");

    // define the image effects contexts we can be used in, in this case a simple filter
    gPropertySuite->propSetString(effectProps,
                                  kOfxImageEffectPropSupportedContexts,
                                  0,
                                  kOfxImageEffectContextFilter);

    return kOfxStatOK;
}
```

You will see that it fetches a property set (via the image effect suite) and sets various properties on it. Specifically the label used in any user interface to name the plugin, and the group of plugins it belongs to. The grouping name allows a developer to ask the host to arrange all plugins with that group name into a single menu/container in the user interface.

The final thing it sets is the single context it can be used in. Contexts are specific to image effect plugins, and they are there because a plugin can be used in many different ways. We call each way an image effect plugin can be used a context. In our example we are saying our plugin can behave as a filter only. A filter is simply an effect with one and only one input clip and one mandated output clip. This is typical of systems such as editors which can drop effects directly onto a clip in a time-line. For more complex systems, e.g. a node graph compositor, you might want to allow the same plugin to have more input clips and a richer parameter set, which we call the general context. A plugin can work one or more contexts, not all of which need be supported by a host.

Because it can be used in different contexts, and will need to be described differently in each, an image effect plugin has a two tier description process. First *kOfxActionDescribe* is called to set attributes common to all the contexts the plugin can be used in, then the *kOfxImageEffectActionDescribeInContext* action is called, once for each context that the host wants to use the effect in.

Again from our example plugin, here is how it responds to the describe in context action...

Note: A plugin developer might package multiple plugins in a single binary and another multiple plugins into multiple

binaries yet both expect them to show up in the same plugin group⁴ in the user interface.

basics.cpp

```
OfxStatus
DescribeInContextAction(OfxImageEffectHandle descriptor, OfxPropertySetHandle inArgs)
{
    // check state
    ERROR_ABORT_IF(gDescribeCalled == false, "DescribeInContextAction called before.
↳ DescribeAction");
    gDescribeInContextCalled = true;

    // get the context from the inArgs handle
    char *context;
    gPropertySuite->propGetString(inArgs, kOfxImageEffectPropContext, 0, &context);

    ERROR_IF(strcmp(context, kOfxImageEffectContextFilter) != 0, "DescribeInContextAction.
↳ called on unsupported contex %s", context);

    OfxPropertySetHandle props;
    // define the mandated single output clip
    gImageEffectSuite->clipDefine(descriptor, "Output", &props);

    // set the component types we can handle on our output
    gPropertySuite->propSetString(props, kOfxImageEffectPropSupportedComponents, 0,
↳ kOfxImageComponentRGBA);
    gPropertySuite->propSetString(props, kOfxImageEffectPropSupportedComponents, 1,
↳ kOfxImageComponentAlpha);

    // define the mandated single source clip
    gImageEffectSuite->clipDefine(descriptor, "Source", &props);

    // set the component types we can handle on our main input
    gPropertySuite->propSetString(props, kOfxImageEffectPropSupportedComponents, 0,
↳ kOfxImageComponentRGBA);
    gPropertySuite->propSetString(props, kOfxImageEffectPropSupportedComponents, 1,
↳ kOfxImageComponentAlpha);

    return kOfxStatOK;
}
```

In this case I've left the error check cluttering up the snippet so you can see how the `inArgs` property set is used to specify which context is currently being described. Our example then goes on to define two image clips, the first used for output, and the second used for input. The API docs specify that a filter effect needs to specify both of these with exactly those names. Not also how the effect is setting a multidimensional property associated with each clip to specify what pixel types it supports on those clips.

For more complex effects, these actions are the point where you specify parameters that the effect wants to use, and get to tweak a whole range of settings to say how the plugin behaves.

⁴ as specified by `kOfxImageEffectPluginPropGrouping`

2.13.3 Creating Instances

So far a host knows what our plugin looks like and how it should behave, but it isn't using it to process pixels yet. At some point a user will click on a button in a UI and to say they want to use the plugin. To do that a host creates an *instance* of the plugin. An instance represents a unique copy of the plugin and contains all the state needed for that. For example, a blur plugin may be instantiated many times in a compositing graph, each instance will have parameters set to a different value, and be connected to different input and output clips.

A plugin developer may need to attach data to each plugin instance, typically to tie the plugin into their own image processing infrastructure. They get the chance to do that via the `kOfxActionCreateInstance` action. The host will call that action just after they have created and initialised their host-side data structures that represent the plugin. Our example plugin doesn't actually do anything on create instance, but it could choose to attached it's own data structures to the instance via the `kOfxPropInstanceData` property.

A plugin will also want to destroy any of its own data structures when an instance is destroyed. It gets to do that in the `kOfxActionDestroyInstance` action.

Our example plugin exercises both of those action just to illustrate what is going it. It simply places a string into the instance data property which it later fetches and destroys. In real plugins, this is typically a hook to deeper plugin side data structures.

Note: Because a host might have asynchronous UI handling and multiple render threads on the same instance, it is suggested that a plugin that wants to write to the instance data after instance creation do so in a safe manner (e.g. by semaphore lock).

basics.cpp

```
OfxStatus CreateInstanceAction(OfxImageEffectHandle instance)
{
    OfxPropertySetHandle effectProps;
    gImageEffectSuite->getPropertySet(instance, &effectProps);

    // attach some instance data to the effect handle, it can be anything
    char *myString = strdup("This is random instance data that could be anything you want.
↵");

    // set my private instance data
    gPropertySuite->propSetPointer(effectProps,
                                  kOfxPropInstanceData,
                                  0,
                                  (void *) myString);

    return kOfxStatOK;
}

// instance destruction
OfxStatus DestroyInstanceAction(OfxImageEffectHandle instance)
{
    OfxPropertySetHandle effectProps;
    gImageEffectSuite->getPropertySet(instance, &effectProps);

    // get my private instance data
    char *myString = NULL;
    gPropertySuite->propGetPointer(effectProps,
```

(continues on next page)

(continued from previous page)

```

        kOfxPropInstanceData,
        0,
        (void **) &myString);
ERROR_ABORT_IF(myString == NULL, "Instance data should not be null!");
free(myString);

return kOfxStatOK;
}

```

Note: *kOfxActionDestroyInstance* should always be called when an instance is destroyed, and furthermore all instances need to have had *kOfxActionDestroyInstance* called on them before *kOfxActionUnload* can be called.

2.13.4 What About The Image Processing?

This plugin is pretty much a *hello world* OFX example, it doesn't actually process any images. Normally a host application would call the *kOfxImageEffectActionRender* action when it wants the plugin to render a frame. Our simple plugin gets around processing any images by trapping the *kOfxImageEffectActionIsIdentity* action. This action lets the plugin tell the host application that it currently does nothing to its inputs, for example a blur effect with the blur size of zero. In such a case the host can simply ignore the plugin and use its source images directly. And here is the code that does that:

basics.cpp

```

OfxStatus IsIdentityAction( OfxImageEffectHandle instance,
                          OfxPropertySetHandle inArgs,
                          OfxPropertySetHandle outArgs)
{
    // we set the name of the input clip to pull data from
    gPropertySuite->propSetString(outArgs, kOfxPropName, 0, "Source");
    return kOfxStatOK;
}

```

The plugin is telling the host to pass through an unprocessed image from an input clip, and because plugins can have more than one input it needs to tell the host which clip to use. It does that by setting the *kOfxPropName* property on the outargs. It also returns *kOfxStatOK* to indicate that it has trapped the action and that the plugin is currently doing nothing.

Remember we said that each action has a well defined set of in and out arguments? In the case of the is identity action these are...

- *kOfxPropTime* - the time at which to test for identity
- *kOfxImageEffectPropFieldToRender* - the field to test for identity
- *kOfxImageEffectPropRenderWindow* - the window to test for identity under
- *kOfxImageEffectPropRenderScale* - the scale factor being applied to the images being rendered
- *kOfxPropName* this to the name of the clip that should be used if the effect is an identity transform, defaults to the empty string
- *kOfxPropTime* the time to use from the indicated source clip as an identity image (allowing time slips to happen), defaults to the value in *kOfxPropTime* in inArgs

A proper plugin would examine the `inArgs`, its parameters and see if it is doing anything to its inputs. If it does need to process images it would return `kOfxStatReplyDefault` rather than `kOfxStatOK`.

2.14 Life Cycle of a Plugin

Now we've outlined the basic actions and functions in a plugin, we should clearly specify the calling sequence. Failure to call them in the right sequence will lead to all sorts of undefined behaviour.

Assuming the host has done nothing apart from load the dynamic library that contains plugins and has found the two *bootstrapping symbols* in the plugin, the host should then...

- call `OfxGetNumberOfPlugins` to discover the number of plugins
- call `OfxGetPlugin` for each of the N plugins in the binary and decide if it can use them or not (by looking at APIs and versions)

At this point the code in the binary should have done nothing apart from run those two functions. The host is free to unload the binary at this point without further interaction with the plugin.

If the host decides it wants to use one of the plugins in the binary it must then...

- call the `setHost` function given to it *for that plugin* and pass back an `OfxHost` struct which allows plugins to fetch suites appropriate for the API
- call the `kOfxActionLoad`
- call `kOfxActionDescribe`
- call `kOfxImageEffectActionDescribeInContext` for each context

If the host wants to actually use a plugin, it creates whatever host side data structures are needed then...

- calls `kOfxActionCreateInstance`

When a host wants to get rid of an instance, before it destroys any of its own data structures it calls `kOfxActionDestroyInstance`

When the host wants to be done with the plugin, and before it dynamically unloads the binary it calls `kOfxActionUnload`, all instances *must* have been destroyed before this call.

Once the final `kOfxActionUnload` has been called, even if it doesn't dynamically unload the binary, the host can no longer call the main entry point on that specific plugin until it once more calls `kOfxActionLoad`.

2.14.1 Packaging A Plugin

The compiled code for a plugin is contained in a dynamic library. Plugins are distributed as a directory structure that allows you to add icons and other resources you may need. There is more detailed information in the OFX Programming Reference Guide.

2.15 Summary

This example has shown you the basics of the OFX plugin machinery, the main things it illustrated was...

- the *two bootstrapper functions* exposed by a plugin that start the plugin discovery process,
- the main entry point of a plugin is given *actions* by the host application to do things,
- the plugin gets *suites* from the host to gain access to functions in the host,
- *property sets* are the main way of passing data back and forth across the API,
- image effect plugins are *described* in a two step process,
- *instances are created* when a host wants to use a plugin to do something
- actions must be called in a *certain order* for the API to work cleanly.

This guide will take you through the fundamentals of processing images in OFX. An example plugin will be used to illustrate how it all works and its source can be found in the C++ file `invert.cpp`. This plugin takes an image and inverts it (or rather calculates the complement of each component). Ideally you should have read the guide to the *basic machinery of an OFX plugin* before you read this guide.

2.16 Action Stations!

The invert example is pretty much the most minimal OFX plugin you can write that processes images. It leaves many things at their default settings which means it doesn't have to trap more than four actions in total¹ and set very few switches.

From the source, here is the main entry routine that traps those actions...

`invert.cpp`

```

////////////////////////////////////
// The main entry point function, the host calls this to get the plugin to do things.
OfxStatus MainEntryPoint(const char *action,
                        const void *handle,
                        OfxPropertySetHandle inArgs,
                        OfxPropertySetHandle outArgs)
{
    // cast to appropriate type
    OfxImageEffectHandle effect = (OfxImageEffectHandle) handle;

    OfxStatus returnStatus = kOfxStatReplyDefault;

    if(strcmp(action, kOfxActionLoad) == 0) {
        // The very first action called on a plugin.
        returnStatus = LoadAction();
    }
    else if(strcmp(action, kOfxActionDescribe) == 0) {
        // the first action called to describe what the plugin does
        returnStatus = DescribeAction(effect);
    }
    else if(strcmp(action, kOfxImageEffectActionDescribeInContext) == 0) {

```

(continues on next page)

¹ I won't bother going into the boot strapping boiler plate, if you are interested you can look at the source directly.

(continued from previous page)

```

    // the second action called to describe what the plugin does
    returnStatus = DescribeInContextAction(effect, inArgs);
}
else if(strcmp(action, kOfxImageEffectActionRender) == 0) {
    // action called to render a frame
    returnStatus = RenderAction(effect, inArgs, outArgs);
}

/// other actions to take the default value
return returnStatus;
}

} // end of anonymous namespace

```

It leaves out some of the actions we had in the last example, which were there for only illustrative purposes only. However, it is now trapping one more, the `kOfxImageEffectActionRender` action. Funnily enough, that is the action called to render a frame of output.

2.17 Describing Our Plugin

We have the standard two step description process for this plugin.

invert.cpp

```

////////////////////////////////////
// the plugin's basic description routine
OfxStatus DescribeAction(OfxImageEffectHandle descriptor)
{
    // set some labels and the group it belongs to
    gPropertySuite->propSetString(effectProps,
                                  kOfxPropLabel,
                                  0,
                                  "OFX Invert Example");
    gPropertySuite->propSetString(effectProps,
                                  kOfxImageEffectPluginPropGrouping,
                                  0,
                                  "OFX Example");

    // define the image effects contexts we can be used in, in this case a simple filter
    gPropertySuite->propSetString(effectProps,
                                  kOfxImageEffectPropSupportedContexts,
                                  0,
                                  kOfxImageEffectContextFilter);

    // set the bit depths the plugin can handle
    gPropertySuite->propSetString(effectProps,
                                  kOfxImageEffectPropSupportedPixelDepths,
                                  0,
                                  kOfxBitDepthFloat);
    gPropertySuite->propSetString(effectProps,
                                  kOfxImageEffectPropSupportedPixelDepths,

```

(continues on next page)

(continued from previous page)

```

        1,
        kOfxBitDepthShort);
gPropertySuite->propSetString(effectProps,
        kOfxImageEffectPropSupportedPixelDepths,
        2,
        kOfxBitDepthByte);

// get the property set handle for the plugin
OfxPropertySetHandle effectProps;
gImageEffectSuite->getPropertySet(descriptor, &effectProps);

// say that a single instance of this plugin can be rendered in multiple threads
gPropertySuite->propSetString(effectProps,
        kOfxImageEffectPluginRenderThreadSafety,
        0,
        kOfxImageEffectRenderFullySafe);

// say that the host should manage SMP threading over a single frame
gPropertySuite->propSetInt(effectProps,
        kOfxImageEffectPluginPropHostFrameThreading,
        0,
        1);

return kOfxStatOK;
}

```

The function called for the describe action sets all the properties on an effect that are independent of specific contexts. In this case it sets some labels and says what contexts it can be used in, which is only the **filter** context, where an effect has a single input and output. It also says what data types it can support when processing images. This is a property that belongs to the plugin as a whole, not to individual clips (see below). If a plugin doesn't support a data type needed by the host, the host is at liberty to ignore it and get on with its life.

We said our plugin supports all the three standard pixel data types, which various properties throughout the API use. The values are:

- *kOfxBitDepthByte* Each component will be an 8 bit unsigned integer with a maximum value of 255.
- *kOfxBitDepthShort* Each component will be an 16 bit unsigned integer with a maximum value of 65535.
- *kOfxBitDepthFloat* Each component will be a 32 bit floating point number with a nominal white point of 1.

Note: The *OfxImageEffectHandle* passed to the describe calls should not be cached away. It only represents some object used while describing the effect. It is *not* the effect itself and when instances are created the handle will refer to a different object entirely. In general, never hang onto any effect handles in any global state.

Finally our plugin is setting some flags to do with multithreaded rendering. The first flag, *kOfxImageEffectPluginRenderThreadSafety* is used to indicate how plugins and instances should be used when rendering in multiple threads. We are setting it to *kOfxImageEffectRenderFullySafe*, which means that the host can have any number of instances rendering and each instance could have possibly have simultaneous renders called on it. (eg: at separate frames). The other options are listed in the programming reference.

The second call sets the *kOfxImageEffectPluginPropHostFrameThreading*, which says that the host should manage any symmetric multiprocessing when rendering the effect. Typically done by calling render on different tiles of the output image. If not set, it is up to the plugin to launch the appropriate number of threads and divide the processing

appropriately across them.

invert.cpp

```

////////////////////////////////////
// describe the plugin in context
OfxStatus
DescribeInContextAction(OfxImageEffectHandle descriptor,
                        OfxPropertySetHandle inArgs)
{
    OfxPropertySetHandle props;
    // define the mandated single output clip
    gImageEffectSuite->clipDefine(descriptor, "Output", &props);

    // set the component types we can handle on our output
    gPropertySuite->propSetString(props,
                                  kOfxImageEffectPropSupportedComponents,
                                  0,
                                  kOfxImageComponentRGBA);
    gPropertySuite->propSetString(props,
                                  kOfxImageEffectPropSupportedComponents,
                                  1,
                                  kOfxImageComponentAlpha);
    gPropertySuite->propSetString(props,
                                  kOfxImageEffectPropSupportedComponents,
                                  2,
                                  kOfxImageComponentRGB);

    // define the mandated single source clip
    gImageEffectSuite->clipDefine(descriptor, "Source", &props);

    // set the component types we can handle on our main input
    gPropertySuite->propSetString(props,
                                  kOfxImageEffectPropSupportedComponents,
                                  0,
                                  kOfxImageComponentRGBA);
    gPropertySuite->propSetString(props,
                                  kOfxImageEffectPropSupportedComponents,
                                  1,
                                  kOfxImageComponentAlpha);
    gPropertySuite->propSetString(props,
                                  kOfxImageEffectPropSupportedComponents,
                                  2,
                                  kOfxImageComponentRGB);

    return kOfxStatOK;
}

```

Here we are describing the plugin when it is being used as a filter. In this case we are describing two clips, the mandated *Source* and *Output* clips. Each clip has a variety of properties on them, in this case we are only setting what pixel components we accept on those inputs. The components supported (unlike the data type) is a per clip thingy. Pixels in OFX can currently only be of three types, which are listed below.

kOfxImageComponentRGBA Each pixel has four samples, corresponding to Red, Green, Blue and Alpha. Packed as RGBA

kOfxImageComponentRGB Each pixel has three samples, corresponding to Red, Green and Blue. Packed as RGB.

kOfxImageComponentAlpha Each pixel has one sample, generally interpreted as an Alpha value.

Note: The OpenGL rendering extension has significantly different set of capabilities for this.

2.18 Clips

I hear you ask “What are these clips of which you speak Mr Nicoletti?”, well they are a sequence of images that vary over time. They are represented in the API by an *OfxImageClipHandle* and have a name plus an associated property set.

Depending on the context, you will have to describe some mandated number of clips with specific names. For example the filter effect has two and only two clips you must describe *Source* and *Output*, a **transition** effect has three and only three clips *SourceFrom*, *SourceTo* and *Output* while a **general** effect has to have one clip called *Output* but as many other input clips as we want. There are **defines** for these in the various OFX header files. The Programming Reference has more information on other contexts, and we will use more in later examples.

There are many properties on a clip, and during description you get to set a whole raft of them as to how you want them to behave. We are relying on the defaults in this example that allow us to avoid issues like field rendering and more.

You fetch images out of clips with a function call in the image effect suite, where you ask for an image at a specific frame. In all cases the clip named “Output” is the one that will give you the images you will be writing to, the other clips are always sources and you should not modify the data in them.

2.19 Images In OFX

Before I start talking over the rendering in the example plugin, I should tell you about images in OFX.

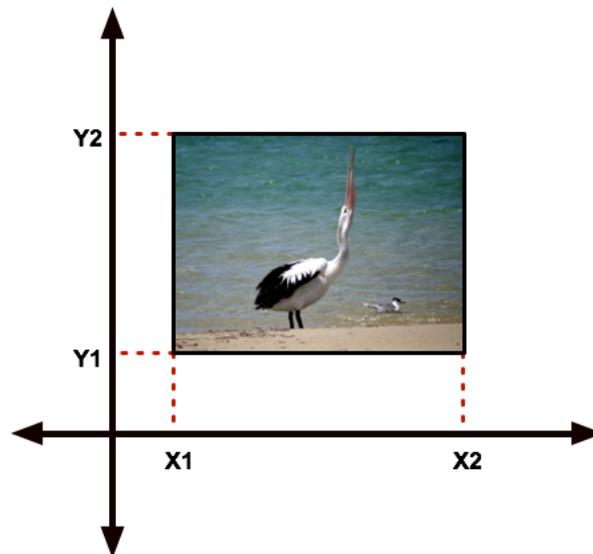
2.19.1 Images and the Image Plane

Images are contiguous rectangular regions of a nominally infinite 2D image plane for which the host has data samples, in the form of *pixels*.

The figure above shows our image spanning the plane from coordinates X1 to X2 in the X dimension and Y1 to Y2 in the Y dimension. We call these four numbers the image’s **bounds**, and is the region an image is guaranteed to have addressable data for.

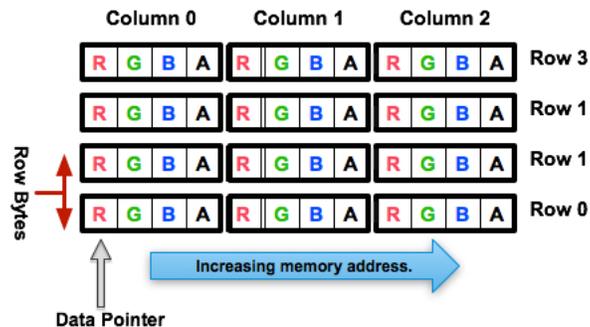
Note: Y goes **up** in OFX land, not down as is common in desktop publishing.

Note: That the image bound is open on the right, so iteration is for (`int x = x1; x < x2; ++x`). This means the number of pixels in the X dimension is given by X2-X1, similarly for the Y dimension.



2.19.2 Image Data

Images are made up of chunk of memory which is interpreted to be a 2D array of pixels. Each pixel in an image has exactly the same number of **components**, each component being of exactly the same **data type**. OFX currently has pixels with one (A), three (RGB) or four components (RGBA), which can be bytes, shorts, or a 32 bit floats.



The figure above shows a small (3x4) image containing RGBA pixels. OFX returns a `void *` data pointer to the first component of the bottom left pixel in the image, which will be at (X1, Y1) on the image plane. Memory addresses increase left to right across the row of an OFX image, with all components and pixels hard packed and contiguous within that row.

Rows may or may not be contiguous in memory, so in our example the address of component **R** at row 1 column 0, may or may not come directly after component **A** at (2, 0). To manage this we use “row bytes”, which are the byte offset between rows, (**not** pixel or component offsets). By breaking this offset out, hosts can more easily map their pixel data into OFX images without having to copy. For example a host that natively runs with Y down and packs images with the top row first in memory would use negative row bytes and have the data pointer point to it’s last row (which is the bottom row).

2.19.3 Pixel Address Calculation

So, given a coordinate on the image plane how do you calculate the address of a pixel in the image? Well you use the following information:

- a `void*` pointer to the bottom left corner of the image
- four integers that define the **bounds** of the image for which there is data
- the data type of each component
- the type of each pixel (which yields the number of components per pixel)
- the number of bytes that is the offset between rows

The code snippet below shows you how to use all that to find the address of a pixel whose coordinates are on the image plane.

invert.cpp

```
// Look up a pixel in the image. returns null if the pixel was not
// in the bounds of the image
template <class T>
static inline T * pixelAddress(int x, int y,
                              void *baseAddress,
                              OfxRectI bounds,
                              int rowBytes,
                              int nCompsPerPixel)
{
    // Inside the bounds of this image?
    if(x < bounds.x1 || x >= bounds.x2 || y < bounds.y1 || y >= bounds.y2)
        return NULL;

    // turn image plane coordinates into offsets from the bottom left
    int yOffset = y - bounds.y1;
    int xOffset = x - bounds.x1;

    // Find the start of our row, using byte arithmetic
    void *rowStartAsVoid = reinterpret_cast<char *>(baseAddress) + yOffset * rowBytes;

    // turn the row start into a pointer to our data type
    T *rowStart = reinterpret_cast<T *>(rowStartAsVoid);

    // finally find the position of the first component of column
    return rowStart + (xOffset * nCompsPerPixel);
}
```

You will notice it is a templated function, where **T** will be instantiated with the appropriate component type by other code. **T** will be one of unsigned `char`, unsigned `short` or `float`.

In order the function...

- checks if the pixel coordinate is within the bounds of the image. If it is not then we have no addressable pixel data at the point, so the function gives up and return `NULL` as an indication of that,
- as we have `x` and `y` as coordinates on the *image plane*, it then turn the coordinates into offsets from the bottom left of the image with a simple subtraction,
- it then finds the start of the row we are interested in by scaling our local `y` offset by `rowBytes` to figure the offset from our base address data pointer, *in bytes*. It adds that to the base address and now has the start of our row.

- it turns the raw address at the start of the row into a pointer of our data type,
- finally it offsets to the correct column by skipping over *xLocal* number of pixels, each of each which contain *nComponents*.

2.19.4 Images Are Property Sets

Images are property sets, you access all the data needed via the standard OFX property mechanism. This has allowed us to expand the information in an image and be 100% backwards compatible to existing hosts and plugins.

Anyway, here is code from our example using the property mechanism to get the required data from an image...

invert.cpp

```

template <class T, int MAX>
void PixelProcessing(OfxImageEffectHandle instance,
                   OfxPropertySetHandle sourceImg,
                   OfxPropertySetHandle outputImg,
                   OfxRectI renderWindow,
                   int nComps)
{
...
    // fetch output image info from the property handle
    int dstRowBytes;
    OfxRectI dstBounds;
    void *dstPtr = NULL;
    gPropertySuite->propGetInt(outputImg, kOfxImagePropRowBytes, 0, &dstRowBytes);
    gPropertySuite->propGetIntN(outputImg, kOfxImagePropBounds, 4, &dstBounds.x1);
    gPropertySuite->propGetPointer(outputImg, kOfxImagePropData, 0, &dstPtr);
...
}

OfxStatus RenderAction( OfxImageEffectHandle instance,
                       OfxPropertySetHandle inArgs,
                       OfxPropertySetHandle outArgs)
{
...
    // figure out the component type
    char *cstr;
    gPropertySuite->propGetString(outputImg, kOfxImageEffectPropComponents, 0, &cstr);
    std::string components = cstr;
...
    // figure out the data types
    gPropertySuite->propGetString(outputImg, kOfxImageEffectPropPixelDepth, 0, &cstr);
    std::string dataType = cstr;
...
}

```

There are many more properties in an image, but we won't need them for this simple example and they'll be covered in other tutorials.

2.20 The Render Action

As stated above, the render action is the one used to get a plugin to actually process images. I'll go through it in stages rather than have one big listing.

invert.cpp

```

////////////////////////////////////
// Render an output image
OfxStatus RenderAction( OfxImageEffectHandle instance,
                        OfxPropertySetHandle inArgs,
                        OfxPropertySetHandle outArgs)
{
    // get the render window and the time from the inArgs
    OfxTime time;
    OfxRectI renderWindow;
    OfxStatus status = kOfxStatOK;

    gPropertySuite->propGetDouble(inArgs, kOfxPropTime, 0, &time);
    gPropertySuite->propGetIntN(inArgs, kOfxImageEffectPropRenderWindow, 4, &renderWindow.
->x1);

```

This first listing shows how the **inArgs** are being used to say what exactly to render. The property *kOfxPropTime* on **inArgs** is the frame of the output clip to render. The property *kOfxImageEffectPropRenderWindow* is the region that should be written to.

The output image (which will be fetched later on) will have a **bounds** that are at least as big as the render window. The bounds of the output image could in fact be larger. This could happen if a host is simultaneously calling the render action in separate threads to perform symmetric multi-processing, each thread would be given a different render window to fill in of the larger output image.

Note: A plugin can have multiple actions being simultaneously in separate threads, especially the render action. Do not rely on any local state if you can help it. You can control how threading works in the describe actions.

Note: To allow a plugin to be called in an SMP manner, or have multiple instances simultaneously rendering, the API has been designed so that the plugin does not rely on any implicit state, such as time, everything is explicit.

invert.cpp

```

// fetch output clip
OfxImageClipHandle outputClip;
gImageEffectSuite->clipGetHandle(instance, "Output", &outputClip, NULL);

// fetch main input clip
OfxImageClipHandle sourceClip;
gImageEffectSuite->clipGetHandle(instance, "Source", &sourceClip, NULL);

```

This next snippet fetches two clip handles by name from the instance, using the image effect suite.²

invert.cpp

² The **NULL** at the end could have been the address of a property set handle if the effect needed to enquire about the clips properties.

```

// the property sets holding our images
OfxPropertySetHandle outputImg = NULL, sourceImg = NULL;
try {
    // fetch image to render into from that clip
    OfxPropertySetHandle outputClip;
    if(gImageEffectSuite->clipGetImage(outputClip, time, NULL, &outputImg) != kOfxStatOK) {
        throw " no output image!";
    }

    // fetch image at render time from that clip
    if (gImageEffectSuite->clipGetImage(sourceClip, time, NULL, &sourceImg) != kOfxStatOK)
    ←{
        throw " no source image!";
    }
}

```

We now (inside a try/catch block) fetch two images from the clips, again using the image effect suite. Note we are asking for images at the frame we were told to render. Effects that need images from other frames can pass in different values to `OfxImageEffectSuiteV1::clipGetImage()`, but will need to trap more actions than we have to make that all work correctly.

We will be given back two property set handles which represent our images. If the call failed (which could be for a variety of good reasons) we give up with a `throw`.

invert.cpp

```

// figure out the data types
char *cstr;
gPropertySuite->propGetString(outputImg, kOfxImageEffectPropComponents, 0, &cstr);
std::string components = cstr;

// how many components per pixel?
int nComps = 0;
if(components == kOfxImageComponentRGBA) {
    nComps = 4;
}
else if(components == kOfxImageComponentRGB) {
    nComps = 3;
}
else if(components == kOfxImageComponentAlpha) {
    nComps = 1;
}
else {
    throw " bad pixel type!";
}

```

Now we want to know what's inside our image's pixels, so we can correctly process it. We ask what components are present in the output image. Because we have left certain settings at the default, the source and output images will always have the same number of components and the same data types. Which is why we aren't checking for the source for its pixel information.

invert.cpp

```

// now do our render depending on the data type
gPropertySuite->propGetString(outputImg, kOfxImageEffectPropPixelDepth, 0, &cstr);
std::string dataType = cstr;

```

(continues on next page)

(continued from previous page)

```

if(dataType == kOfxBitDepthByte) {
    PixelProcessing<unsigned char, 255>(instance, sourceImg, outputImg, renderWindow,
    ↪nComps);
}
else if(dataType == kOfxBitDepthShort) {
    PixelProcessing<unsigned short, 65535>(instance, sourceImg, outputImg, renderWindow,
    ↪nComps);
}
else if (dataType == kOfxBitDepthFloat) {
    PixelProcessing<float, 1>(instance, sourceImg, outputImg, renderWindow, nComps);
}
else {
    throw " bad data type!";
    throw 1;
}

```

Now we are enquiring as to what C type the components our image will be. Again throwing if something has gone wrong. We use the data type to correctly instantiate our templated function which will do the grunt work of iterating over pixels. Note also that it is passing the nominal maximum value of the data type as a template argument.

invert.cpp

```

}
catch(const char *errStr ) {
    bool isAborting = gImageEffectSuite->abort(instance);

    // if we were interrupted, the failed fetch is fine, just return kOfxStatOK
    // otherwise, something weird happened
    if(!isAborting) {
        status = kOfxStatFailed;
    }
    ERROR_IF(!isAborting, " Rendering failed because %s", errStr);
}

if(sourceImg)
    gImageEffectSuite->clipReleaseImage(sourceImg);
if(outputImg)
    gImageEffectSuite->clipReleaseImage(outputImg);

// all was well
return status;
}

```

This last bit is basically clean up. We have the catch for our try/catch block. The first thing it does is ask the host application is the effect being told to stop by calling the `OfxImageEffectSuiteV1::abort()` function on the effect suite. We might have ended up in the catch block because the an image could not be fetched, if that was a side effect of the host interrupting processing, it is *not* counted as an error. So we check that before we return a failed error state from our action.

Finally we release the images we have fetched and return the error status.

Note: Images should not be held onto outside the scope of the action they were fetched in, the data will not be guaranteed to be valid. It is polite to release them as soon as possible, especially if you are fetching multiple images on input.

Now for our pixel pushing code.³

invert.cpp

```
// iterate over our pixels and process them
template <class T, int MAX>
void PixelProcessing(OfxImageEffectHandle instance,
                   OfxPropertySetHandle sourceImg,
                   OfxPropertySetHandle outputImg,
                   OfxRectI renderWindow,
                   int nComps)
{
    // fetch output image info from the property handle
    int dstRowBytes;
    OfxRectI dstBounds;
    void *dstPtr = NULL;
    gPropertySuite->propGetInt(outputImg, kOfxImagePropRowBytes, 0, &dstRowBytes);
    gPropertySuite->propGetIntN(outputImg, kOfxImagePropBounds, 4, &dstBounds.x1);
    gPropertySuite->propGetPointer(outputImg, kOfxImagePropData, 0, &dstPtr);

    if(dstPtr == NULL) {
        throw "Bad destination pointer";
    }

    // fetch input image info from the property handle
    int srcRowBytes;
    OfxRectI srcBounds;
    void *srcPtr = NULL;
    gPropertySuite->propGetInt(sourceImg, kOfxImagePropRowBytes, 0, &srcRowBytes);
    gPropertySuite->propGetIntN(sourceImg, kOfxImagePropBounds, 4, &srcBounds.x1);
    gPropertySuite->propGetPointer(sourceImg, kOfxImagePropData, 0, &srcPtr);

    if(srcPtr == NULL) {
        throw "Bad source pointer";
    }
}
```

We've shown bits of this before. Here we have a templated function that we use to process our pixels. It is templated on the data type that the components in each pixel will be, as well as a nominal *max* value to use in our invert computation.

The first thing it does is to pull out the bounds, rowbytes and destination pointer of our two images. We can now iterate over the render window and set pixels in the output image.

invert.cpp

```
// and do some inverting
for(int y = renderWindow.y1; y < renderWindow.y2; y++) {
    if(y % 20 == 0 && gImageEffectSuite->abort(instance)) break;
```

(continues on next page)

³ This is purely illustrative as to how the API works, it is in no way fast code, I would be ashamed to put code like this into a serious piece of image processing.

```

// get the row start for the output image
T *dstPix = pixelAddress<T>(renderWindow.x1, y, dstPtr, dstBounds, dstRowBytes,
↪nComps);

for(int x = renderWindow.x1; x < renderWindow.x2; x++) {

    // get the source pixel
    T *srcPix = pixelAddress<T>(x, y, srcPtr, srcBounds, srcRowBytes, nComps);

    if(srcPix) {
        // we have one, iterate each component in the pixels
        for(int i = 0; i < nComps; ++i) {
            if(i != 3) { // We don't invert alpha.
                *dstPix = MAX - *srcPix; // invert
            }
            else {
                *dstPix = *srcPix;
            }
            ++dstPix; ++srcPix;
        }
    }
    else {
        // we don't have a pixel in the source image, set output to black
        for(int i = 0; i < nComps; ++i) {
            *dstPix = 0;
            ++dstPix;
        }
    }
}
}
}
}

```

The first thing we do at each row we are processing is to check that the host hasn't told our plugin to abort processing. (Ideally you can do this a bit less often than every line). We only do this every 20th row, as the overhead on the host side to check for an abort might be quite high.

The next thing we do is to use the `pixelAddress` function to find the address of the first component of the first pixel in the current, and we put it in `dstPix`. Because we have a guarantee that the bounds of the output image are at least as big as the render window, we can simply increment `dstPix` across the row as we iterate over the image.

Now we iterate across the row. We attempt to fetch the address of the source pixel at our `x,y` location in the image plane. If we get it we iterate over the number of component, setting the output to be the invert⁴ of the input. If we don't get it, we set the output pixel to all zero.

Note: You notice that we are continually calculating the address of `srcPix` at each pixel location and not incrementing the pointer as we could with `dstPix`. The reason for this is that, at the default settings, there is no guarantee as to the bounds of the input image. It need not be congruent with any other input, the output or the render window.

I could obviously write this much more efficiently and avoid the continual address calculation. However for illustrative purposes I haven't done that.

⁴ complement really

2.21 Summary

This plugin has shown you the basics of working with OFX images, the main things it illustrated were...

- what are *clips* and how we get images from clips,
- how *images* are laid out in memory and how to access pixels,
- the basics of the *render action*

This guide will take you through the basics of creating and using parameters in OFX. An example plugin will be used to illustrate how it all works and its source can be found in the C++ file `gain.cpp`. This plugin takes an image and multiplies the pixel by the value held in a user visible parameter. Ideally you should have read the guide to the *basic image processing* before you read this guide.

2.22 Parameters

Host applications need parameters to make their own effects work. Such as the size of a blur effect, colours to apply to some text, a point for the centre of a lens flare and so on. The host app will use some widget set to present a user interface for the parameter, have ways to do undo/redo, saving/loading, manage animation etc...

The OFX parameters suite is the bridge from a plugin to the host's native parameter set. So plugin devs don't have to do all that work for themselves and also get the other advantages a host's parameters system may give (e.g. scripting).

The way it works is fairly simple in concept, we get a plugin to tell the host what parameters it wants during description. When an instance is created, the host will make whatever native data structures it needs to manage those params. The plugin can then grab values from the various parameters to do what it needs to do during various actions, it can even write back to parameters under certain conditions.

Our simple gain example will make two parameters, a parameter that is of type `double`¹ which is the gain amount and a `bool` param which controls whether to apply the gain amount to the Alpha of an RGBA image. There are more parameter types, and quite a few properties you can set on a param to control how it should behave in a host application. The current parameter types available are *listed here*.

Note: A key concept in OFX is that the state of a plugin instance is totally and uniquely defined by the value of its parameters and input clips. You are asking for trouble if you try and store important data in separate files and so on. The host won't be able to properly know when things have changed, how to manage that extra data and so on. Attempting to manage data outside of the parameter set will almost certainly cause hosts to lose track of the correct render and you will get angry users. This is a fundamental aspect of the API. If it isn't in a parameter, it is going to cause problems with the host if you rely on it.

¹ the API manages all floating point params as doubles, the host could be using 32 bit floats, or fixed precision for that matter, so long as the values are passed back and forth over the API as doubles, all will be fine

2.23 Actions

This example doesn't trap any actions you haven't already seen in the other examples, it just does a little bit more in them. Seeing as you should be familiar with how the main entry point works, I won't bother with the code listing from now on. The actions our plugin traps are now:

- *kOfxActionLoad* - to grab suites from the host,
- *kOfxActionDescribe* and *kOfxImageEffectActionDescribeInContext* - to describe the plugin to the host, *including parameters*,
- *kOfxActionCreateInstance* and *kOfxActionDestroyInstance* - to create and destroy instance data, where we cache handles to clips and parameters,
- *kOfxImageEffectActionIsIdentity* - to check if the parameter values are at their defaults and so the plugin can be ignore by the host,
- *kOfxImageEffectActionRender* - to actually process pixels.

Now seeing as we are going to be playing with parameters, our plugin will need a new suite, the parameters suite, and our load action now looks like:

gain.cpp

```
OfxPropertySuiteV1 *gPropertySuite = 0;
OfxImageEffectSuiteV1 *gImageEffectSuite = 0;
OfxParameterSuiteV1 *gParameterSuite = 0;

////////////////////////////////////
// get the named suite and put it in the given pointer, with error checking
template <class SUITE>
void FetchSuite(SUITE *& suite, const char *suiteName, int suiteVersion)
{
    suite = (SUITE *) gHost->fetchSuite(gHost->host, suiteName, suiteVersion);
    if(!suite) {
        ERROR_ABORT_IF(suite == NULL,
            "Failed to fetch %s version %d from the host.",
            suiteName,
            suiteVersion);
    }
}

////////////////////////////////////
// The first _action_ called after the binary is loaded
OfxStatus LoadAction(void)
{
    // fetch our three suites
    FetchSuite(gPropertySuite, kOfxPropertySuite, 1);
    FetchSuite(gImageEffectSuite, kOfxImageEffectSuite, 1);
    FetchSuite(gParameterSuite, kOfxParameterSuite, 1);

    return kOfxStatOK;
}
```

You can see I've written a `FetchSuite` function, as I got bored of writing the same code over and over. We are now fetching the a suite of type *OfxParameterSuiteV1* which is defined in the header file *ofxParam.h*.²

² The suite is completely independent of the image effect suite and could happily be used to describe parameters to other types of plugins

2.24 Describing Our Plugin

We have the standard two step description process for this plugin. The Describe action is almost exactly the same as in our previous examples, some names and labels have been changed is all, so I won't list it. However, the describe in context action has a few more things going on.

In the listings below I've chopped out the code to describe clips, as it is exactly the same as in the last example. What's new is the bit where we describe parameters. I'll show the describe in context action in several small chunks to take you through it.

gain.cpp

```
OfxStatus
DescribeInContextAction(OfxImageEffectHandle descriptor,
                        OfxPropertySetHandle inArgs)
{
    ...
    BIG SNIP OF EXACTLY THE SAME CODE IN THE LAST EXAMPLE
    ...

    // first get the handle to the parameter set
    OfxParamSetHandle paramSet;
    gImageEffectSuite->getParamSet(descriptor, &paramSet);

    // properties on our parameter
    OfxPropertySetHandle paramProps;

    // now define a 'gain' parameter and set its properties
    gParameterSuite->paramDefine(paramSet,
                                kOfxParamTypeDouble,
                                GAIN_PARAM_NAME,
                                &paramProps);
```

The first thing we do is to grab a `OfxParamSetHandle` from the effect descriptor. This object represents all the parameters attached to a plugin and is independent and orthogonal to an image effect.

The parameter suite is then used to define a parameter on that parameter set. In this case its type is double, and its name is "gain". These are the two most important things for a parameter.

Note: The name uniquely identifies that parameter within the API, so no two parameters can have the same name.

The last argument to `paramDefine` is an optional pointer to the new parameter's property set handle. Each parameter has a set of properties we use to refine its behaviour, most of which have sensible defaults.

gain.cpp

```
gPropertySuite->propSetString(paramProps,
                              kOfxParamPropDoubleType,
                              0,
                              kOfxParamDoubleTypeScale);
```

The first property on our `gain` param we set is the kind of double parameter it is. Many host applications have different kind of double parameters and user interfaces that make working with them easier. For example a parameter used to control a rotation might have a little dial in the UI to spin the angle, a 2D position parameter might get cross hairs over

the image and so on. In this case we are saying that our double parameter represents a scaling value. OFX has more kinds of double parameter which you can use to best for your effect.

gain.cpp

```
gPropertySuite->propSetDouble(paramProps,
                              kOfxParamPropDefault,
                              0,
                              1.0);
gPropertySuite->propSetDouble(paramProps,
                              kOfxParamPropMin,
                              0,
                              0.0);
```

This section sets a default value for our parameter and a logical a minimum value below which it cannot go. Note it does not set a maximum value, so the parameter should not be clamped to any upper value ever.

gain.cpp

```
gPropertySuite->propSetDouble(paramProps,
                              kOfxParamPropDisplayMin,
                              0,
                              0.0);
gPropertySuite->propSetDouble(paramProps,
                              kOfxParamPropDisplayMax,
                              0,
                              10.0);
```

Numbers are often manipulated with sliders widgets in user interfaces, and it is useful to set a range on those sliders. Which is exactly what we are doing here. This is distinct to the logical minimum and maximum values, so you can set a *useful* range for the UI, but still allow the values to be outside that range. So here a slider would only allow values between 0.0 and 10.0 for our gain param, but the parameter could be set to a million via other means, eg: typing in a UI number box, animation, scripting whatever.

gain.cpp

```
gPropertySuite->propSetString(paramProps,
                              kOfxPropLabel,
                              0,
                              "Gain");
gPropertySuite->propSetString(paramProps,
                              kOfxParamPropHint,
                              0,
                              "How much to multiply the image by.");
```

Here we are setting two text field on the param. The first is a label for the parameter. This is to be used in any UI the host has to label the parameter. It defaults to the name of the param, but it can be entirely different. Finally we set a hint string to be used for the parameter.

gain.cpp

```
// and define the 'applyToAlpha' parameters and set its properties
gParameterSuite->paramDefine(paramSet,
                             kOfxParamTypeBoolean,
                             APPLY_TO_ALPHA_PARAM_NAME,
                             &paramProps);
```

(continues on next page)

(continued from previous page)

```

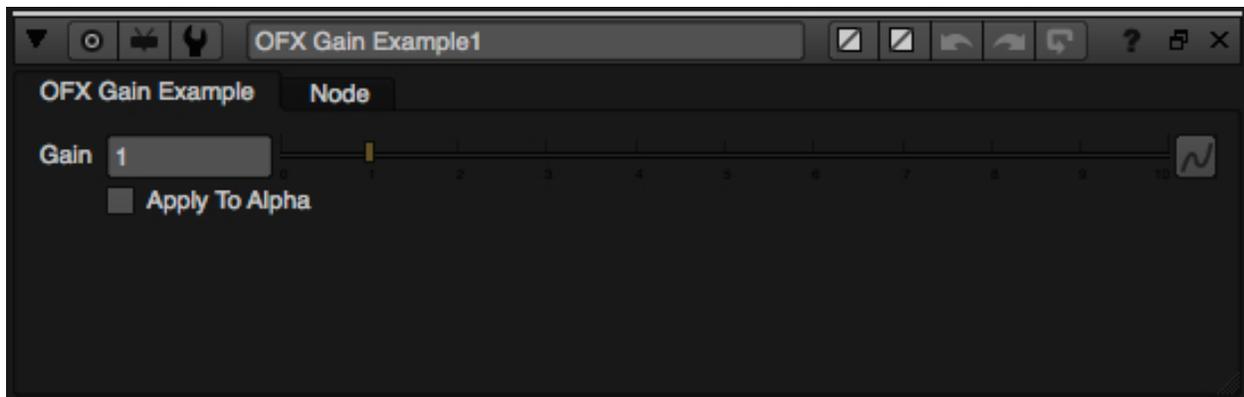
gPropertySuite->propSetInt(paramProps,
                          kOfxParamPropDefault,
                          0,
                          0);
gPropertySuite->propSetString(paramProps,
                              kOfxParamPropHint,
                              0,
                              "Whether to apply the gain value to alpha as well.");
gPropertySuite->propSetString(paramProps,
                              kOfxPropLabel,
                              0,
                              "Apply To Alpha");

return kOfxStatOK;
}

```

In this last section we define a second parameter, named *applyToAlpha*, which is of type boolean. We then set some obvious state on it and we are done. Notice the label we set, it is much clearer to read than the name.

And that's it, we've defined two parameters for our plugin. There are many more properties you can set on your plugin to control how they behave and to give hints as to what you are going to do to them.



Finally, the image above shows the control panel for an instance of our example inside Nuke.

2.25 Instances and Parameters

When the host creates an instance of the plugin, it will first create all the native data structures it needs to represent the plugin, fully populate them with the required values, and only then call the create instance action.

So what happens in the create instance action then? Possibly nothing, you can always grab parameters from an instance by name at any time. But to make our code a bit cleaner and to show an example of instance data being used, we are going to trap create instance.

gain.cpp

```

////////////////////////////////////
// our instance data, where we are caching away clip and param handles
struct MyInstanceData {
    // handles to the clips we deal with

```

(continues on next page)

(continued from previous page)

```

OfxImageClipHandle sourceClip;
OfxImageClipHandle outputClip;

// handles to a our parameters
OfxParamHandle gainParam;
OfxParamHandle applyToAlphaParam;
};

```

To stop duplicating code all over, and to minimise fetches to various handles, we are going to cache away handles to our clips and parameters in a simple struct. Note that these handles are valid for the duration of the instance.

gain.cpp

```

////////////////////////////////////
// instance construction
OfxStatus CreateInstanceAction( OfxImageEffectHandle instance)
{
    OfxPropertySetHandle effectProps;
    gImageEffectSuite->getPropertySet(instance, &effectProps);

    // To avoid continual lookup, put our handles into our instance
    // data, those handles are guaranteed to be valid for the duration
    // of the instance.
    MyInstanceData *myData = new MyInstanceData;

    // Set my private instance data
    gPropertySuite->propSetPointer(effectProps, kOfxPropInstanceData, 0, (void *) myData);

    // Cache the source and output clip handles
    gImageEffectSuite->clipGetHandle(instance, "Source", &myData->sourceClip, 0);
    gImageEffectSuite->clipGetHandle(instance, "Output", &myData->outputClip, 0);

    // Cache away the param handles
    OfxParamSetHandle paramSet;
    gImageEffectSuite->getParamSet(instance, &paramSet);
    gParameterSuite->paramGetHandle(paramSet,
                                    GAIN_PARAM_NAME,
                                    &myData->gainParam,
                                    0);
    gParameterSuite->paramGetHandle(paramSet,
                                    APPLY_TO_ALPHA_PARAM_NAME,
                                    &myData->applyToAlphaParam,
                                    0);

    return kOfxStatOK;
}

```

So here is the function called when we trap a create instance action. You can see that it allocates a MyInstanceData struct and caches it away in the instance's property set.

It then fetches handles to the two clips and two parameters by name and caches those into the newly created struct.

gain.cpp

```

////////////////////////////////////
// get my instance data from a property set handle
MyInstanceData *FetchInstanceData(OfxPropertySetHandle effectProps)
{
    MyInstanceData *myData = 0;
    gPropertySuite->propGetPointer(effectProps,
                                  kOfxPropInstanceData,
                                  0,
                                  (void **) &myData);

    return myData;
}

```

And here is a simple function to fetch instance data. It is actually overloaded and there is another version that take an `OfxImageEffectHandle`.

Of course we now need to trap the destroy instance action to delete our instance data, otherwise we will get memory leaks.

gain.cpp

```

////////////////////////////////////
// instance destruction
OfxStatus DestroyInstanceAction( OfxImageEffectHandle instance)
{
    // get my instance data
    MyInstanceData *myData = FetchInstanceData(instance);
    delete myData;

    return kOfxStatOK;
}

```

2.26 Getting Values From Instances

So we've define our parameters, we've got handles to the instance of them, but we will want to grab the value of the parameters to actually use them at render time.

gain.cpp

```

////////////////////////////////////
// Render an output image
OfxStatus RenderAction( OfxImageEffectHandle instance,
                       OfxPropertySetHandle inArgs,
                       OfxPropertySetHandle outArgs)
{
    // get the render window and the time from the inArgs
    OfxTime time;
    OfxRectI renderWindow;
    OfxStatus status = kOfxStatOK;

    gPropertySuite->propGetDouble(inArgs, kOfxPropTime, 0, &time);
    gPropertySuite->propGetIntN(inArgs, kOfxImageEffectPropRenderWindow, 4, &
    ↪renderWindow.x1);
}

```

(continues on next page)

(continued from previous page)

```

// get our instance data which has out clip and param handles
MyInstanceData *myData = FetchInstanceData(instance);

// get our param values
double gain = 1.0;
int applyToAlpha = 0;
gParameterSuite->paramGetValueAtTime(myData->gainParam, time, &gain);
gParameterSuite->paramGetValueAtTime(myData->applyToAlphaParam, time, &applyToAlpha);
....

```

We are using the `OfxParameterSuiteV1::paramGetValueAtTime()` suite function to get the value of our parameters for the given time we are rendering at. Nearly all actions passed to an instance will have a time to perform the instance at, you should use this when fetching values out of a param.

The param get value functions use var-args to return values to plugins, similar to a C scanf function.

And finally here is a snippet of the templated pixel pushing code where we do the actual processing using our parameter values;

gain.cpp

```

// and do some processing
for(int y = renderWindow.y1; y < renderWindow.y2; y++) {
    if(y % 20 == 0 && gImageEffectSuite->abort(instance)) break;

    // get the row start for the output image
    T *dstPix = pixelAddress<T>(renderWindow.x1, y,
                               dstPtr,
                               dstBounds,
                               dstRowBytes,
                               nComps);

    for(int x = renderWindow.x1; x < renderWindow.x2; x++) {

        // get the source pixel
        T *srcPix = pixelAddress<T>(x, y,
                                    srcPtr,
                                    srcBounds,
                                    srcRowBytes,
                                    nComps);

        if(srcPix) {
            // we have one, iterate each component in the pixels
            for(int i = 0; i < nComps; ++i) {
                if(i != 3 || applyToAlpha) {
                    // multiply our source component by our gain value
                    double value = *srcPix * gain;

                    // if it has gone out of legal bounds, clamp it
                    if(MAX != 1) { // we let floating point pixels over and underflow
                        value = value < 0 ? 0 : (value > MAX ? MAX : value);
                    }
                }
            }
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

        *dstPix = T(value);
    }
    else {
        *dstPix = *srcPix;
    }
    // increment to next component
    ++dstPix; ++srcPix;
}
}
else {
    // we don't have a pixel in the source image, set output to zero
    for(int i = 0; i < nComps; ++i) {
        *dstPix = 0;
        ++dstPix;
    }
}
}
}
}

```

Notice that we are checking to see if `MAX != 1`, which means our pixels are not floating point. If that is the case, we are clamping the pixel's value so we don't get integer overflow.

2.27 Summary

This plugin has shown you the basics of working with OFX parameters, the main things it illustrated were:

- defining parameters in the define in context action,
- setting properties to control the behaviour of parameters,
- using the instance data pointer to cache away handles to instances of parameters and clips,
- fetching values of a parameter from parameter instance handles and using them to process pixels.

This guide will take you through the basics of creating effects that can be used in more than one context, as well as how to make a multi-input effect. Its source can be found in the C++ file `saturation.cpp`. This plugin takes an RGB or RGBA image and increases or decreases the saturation by a parameter. It can be used in two contexts, firstly as a simple filter, secondly as a general effect, where it has an optional second input clip which is used to control where the effect is applied.

2.28 Multiple Contexts, Why Bother?

As briefly described in the first example, OFX has the concept of contexts that an effect can be used in. Our example is going to work in the filter context and the general context.

The rules for a filter context are that it has to have one and only one input clip, called *Source* and one and only one output clip called *Output*.

For a general context, you have to have a single mandated clip called *Output* and that is it. You are free to have as many input clips as you need, name them how you feel and use choose how to set certain important properties of the output.

Why would we want to do this? Because not all host applications behave the same way. For example an editing application will typically allow effects to be applied to clips on a timeline, and the effect can only take a single input

when used like that. A complicated node-based compositor is less restrictive, its effect can typically have any number of inputs and the rules for certain behaviours are relaxed.

So you've written your OFX effect, and it can work with a single input, but would ideally work much better with multiple inputs. You also want it to work as best it can across a range of host applications. If you could only write it as a multi-input general effect with more than one input, it couldn't work in an editor. However if you wrote it as a single input effect, it wouldn't work as well as it could in a node based compositor. Having your effect work in multiple contexts is the way to have it work as best as possible in both applications.

In this way an OFX host application, which knows which contexts it can support, will inspect the contexts a plugin says it can be used in, and choose the most appropriate one for what it wants to do.

This example plugin shows you how to do that.

2.29 Describing Our Plugin

Our basic describe action is pretty much the same as all the other examples, but with one minor difference, we set two contexts in which the effect can be used in.

saturation.cpp

```
// Define the image effects contexts we can be used in, in this case a filter
// and a general effect.
gPropertySuite->propSetString(effectProps,
                             kOfxImageEffectPropSupportedContexts,
                             0,
                             kOfxImageEffectContextFilter);

gPropertySuite->propSetString(effectProps,
                             kOfxImageEffectPropSupportedContexts,
                             1,
                             kOfxImageEffectContextGeneral);
```

The snippet above shows that the effect is saying it can be used in the filter and general contexts.

Both of these have rules associated as to how the plugin behaves in that context. Because the filter context is so simple, most of the default behaviour just works and you don't have to trap many other actions.

In the case of the general context, the default behaviour might not work the way you want, and you may have to trap other actions. Fortunately the defaults work for us as will.

saturation.cpp

```
////////////////////////////////////
// describe the plugin in context
OfxStatus
DescribeInContextAction(OfxImageEffectHandle descriptor,
                       OfxPropertySetHandle inArgs)
{
    // get the context we are being described for
    char *context;
    gPropertySuite->propGetString(inArgs, kOfxImageEffectPropContext, 0, &context);

    OfxPropertySetHandle props;
    // define the mandated single output clip
```

(continues on next page)

(continued from previous page)

```

gImageEffectSuite->clipDefine(descriptor, "Output", &props);

// set the component types we can handle on our output
gPropertySuite->propSetString(props,
                              kOfxImageEffectPropSupportedComponents,
                              0,
                              kOfxImageComponentRGBA);
gPropertySuite->propSetString(props,
                              kOfxImageEffectPropSupportedComponents,
                              1,
                              kOfxImageComponentRGB);

// define the mandated single source clip
gImageEffectSuite->clipDefine(descriptor, "Source", &props);

// set the component types we can handle on our main input
gPropertySuite->propSetString(props,
                              kOfxImageEffectPropSupportedComponents,
                              0,
                              kOfxImageComponentRGBA);
gPropertySuite->propSetString(props,
                              kOfxImageEffectPropSupportedComponents,
                              1,
                              kOfxImageComponentRGB);

if(strcmp(context, kOfxImageEffectContextGeneral) == 0) {
    gImageEffectSuite->clipDefine(descriptor, "Mask", &props);

    // set the component types we can handle on our main input
    gPropertySuite->propSetString(props,
                                  kOfxImageEffectPropSupportedComponents,
                                  0,
                                  kOfxImageComponentAlpha);
    gPropertySuite->propSetInt(props,
                              kOfxImageClipPropOptional,
                              0,
                              1);
    gPropertySuite->propSetInt(props,
                              kOfxImageClipPropIsMask,
                              0,
                              1);
}

...
[SNIP]
...

return kOfxStatOK;
}

```

I've snipped the simple parameter definition code out to save some space.

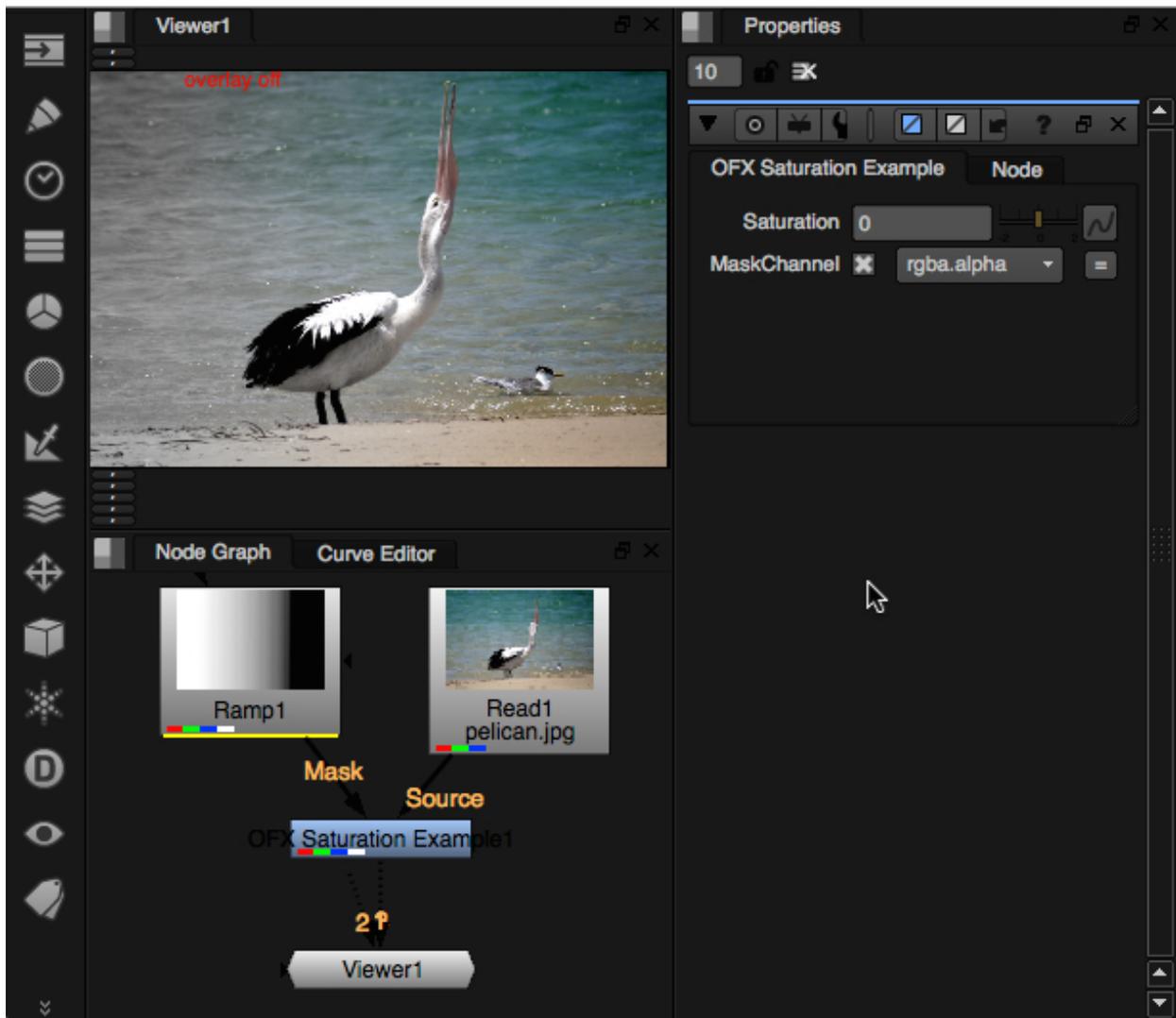
Here we have the describe in context action. This will now be called once for each context that a host application wants

to support. You know which context you are being described in by the `kOfxImageEffectPropContext` property on `inArgs`.

Regardless of the context, it describes two clips, “Source” and “Output”, which will work fine both as a filter and a general context. Note that we won’t support *alpha* on these two clips, we only support images that have colour components, as how can you saturate a single channel image?

Finally, if the effect is in the general context, we describe a third clip and call it “Mask”. We then tell the host about that clip...

- firstly, that we only want single component images from that clip
- secondly, that the clip is optional,
- thirdly, that this clip is to be interpreted as a mask, so hosts that manage such things separately, know it can be fed into this input.



The image above shows our saturation example running inside Nuke. Nuke chose to instantiate the plugin as a general context effect, not a filter, as general contexts are the ones it prefers. You can see the graph, and our saturation node has two inputs, one for the mask and one for the source image. The control panel for the effect is also shown, with the saturation value set to zero. Note the extra *MaskChannel* param, which was not specified by the plugin. This was

automatically generated by Nuke when it saw that the *Mask* input to the effect was a single channel, so as to allow the user to choose which one to use as a mask.

The result is an image whose desaturation amount is modulated by the alpha channel of the mask image, which in this case is a right to left ramp.

2.30 The Other Actions

All the other actions should be fairly familiar and you should be able to reason them out pretty easily. The two that have any significant differences because of the multi context use are the create instance action and the render action.

2.30.1 Create Instance

This is pretty familiar, though we have a slight change to handle the mask input.

saturation.cpp

```

////////////////////////////////////
/// instance construction
OfxStatus CreateInstanceAction( OfxImageEffectHandle instance)
{
    OfxPropertySetHandle effectProps;
    gImageEffectSuite->getPropertySet(instance, &effectProps);

    // To avoid continual lookup, put our handles into our instance
    // data, those handles are guaranteed to be valid for the duration
    // of the instance.
    MyInstanceData *myData = new MyInstanceData;

    // Set my private instance data
    gPropertySuite->propSetPointer(effectProps, kOfxPropInstanceData, 0, (void *) myData);

    // is this instance made for the general context?
    char *context = 0;
    gPropertySuite->propGetString(effectProps, kOfxImageEffectPropContext, 0, &context);
    myData->isGeneralContext = context &&
        (strcmp(context, kOfxImageEffectContextGeneral) == 0);

    // Cache the source and output clip handles
    gImageEffectSuite->clipGetHandle(instance, "Source", &myData->sourceClip, 0);
    gImageEffectSuite->clipGetHandle(instance, "Output", &myData->outputClip, 0);

    if(myData->isGeneralContext) {
        gImageEffectSuite->clipGetHandle(instance, "Mask", &myData->maskClip, 0);
    }

    // Cache away the param handles
    OfiParamSetHandle paramSet;
    gImageEffectSuite->getParamSet(instance, &paramSet);
    gParameterSuite->paramGetHandle(paramSet,
        SATURATION_PARAM_NAME,
        &myData->saturationParam,

```

(continues on next page)

(continued from previous page)

```

        0);

    return kOfxStatOK;
}

```

We are again using instance data to cache away a set of handles to clips and params (the constructor of which sets them all to NULL). We are also recording which context we have had our instance created for by checking the `kOfxImageEffectPropContext` property of the effect. If it is a general context we also cache the `Mask` input in our instance data. Pretty easy.

2.30.2 Rendering

Because we are now using a class to wrap up OFX images (see *below*) the render code is a bit tidier but is pretty much still the same really. The major difference is that we are now fetching a third image, for the mask image, and we are prepared for this to fail and keep going as we may be in the filter context, or we may be in the general context but the clip is not connected.

saturation.cpp

```

// Render an output image
OfxStatus RenderAction( OfxImageEffectHandle instance,
                       OfxPropertySetHandle inArgs,
                       OfxPropertySetHandle outArgs)
{
    // get the render window and the time from the inArgs
    OfxTime time;
    OfxRectI renderWindow;
    OfxStatus status = kOfxStatOK;

    gPropertySuite->propGetDouble(inArgs,
                                  kOfxPropTime,
                                  0,
                                  &time);
    gPropertySuite->propGetIntN(inArgs,
                               kOfxImageEffectPropRenderWindow,
                               4,
                               &renderWindow.x1);

    // get our instance data which has out clip and param handles
    MyInstanceData *myData = FetchInstanceData(instance);

    // get our param values
    double saturation = 1.0;
    gParameterSuite->paramGetValueAtTime(myData->saturationParam, time, &saturation);

    // the property sets holding our images
    OfxPropertySetHandle outputImg = NULL, sourceImg = NULL, maskImg = NULL;
    try {
        // fetch image to render into from that clip
        Image outputImg(myData->outputClip, time);
        if(!outputImg) {
            throw " no output image!";

```

(continues on next page)

(continued from previous page)

```

}

// fetch image to render into from that clip
Image sourceImg(myData->sourceClip, time);
if(!sourceImg) {
    throw " no source image!";
}

// fetch mask image at render time from that clip, it may not be there
// as we might in the filter context or it might not be attached as it
// is optional, so don't worry if we don't have one.
Image maskImg(myData->maskClip, time);

// now do our render depending on the data type
if(outputImg.bytesPerComponent() == 1) {
    PixelProcessing<unsigned char, 255>(saturation,
                                       instance,
                                       sourceImg,
                                       maskImg,
                                       outputImg,
                                       renderWindow);
}
else if(outputImg.bytesPerComponent() == 2) {
    PixelProcessing<unsigned short, 65535>(saturation,
                                           instance,
                                           sourceImg,
                                           maskImg,
                                           outputImg,
                                           renderWindow);
}
else if(outputImg.bytesPerComponent() == 4) {
    PixelProcessing<float, 1>(saturation,
                              instance,
                              sourceImg,
                              maskImg,
                              outputImg,
                              renderWindow);
}
else {
    throw " bad data type!";
    throw 1;
}
}
catch(const char *errStr ) {
    bool isAborting = gImageEffectSuite->abort(instance);

    // if we were interrupted, the failed fetch is fine, just return kOfxStatOK
    // otherwise, something weird happened
    if(!isAborting) {
        status = kOfxStatFailed;
    }
}

```

(continues on next page)

(continued from previous page)

```

    ERROR_IF(!isAborting, " Rendering failed because %s", errStr);
}

// all was well
return status;
}

```

The actual pixel processing code does the standard saturation calculation on each pixel, scaling each of R, G and B around their common average. The tweak we add is to modulate the amount of the effect by looking at the pixel values of the mask input if we have one. Again this is not meant to be fast code, just illustrative.

saturation.cpp

```

////////////////////////////////////
// iterate over our pixels and process them
template <class T, int MAX>
void PixelProcessing(double saturation,
                    OfxImageEffectHandle instance,
                    Image &src,
                    Image &mask,
                    Image &output,
                    OfxRectI renderWindow)
{
    int nComps = output.nComponents();

    // and do some processing
    for(int y = renderWindow.y1; y < renderWindow.y2; y++) {
        if(y % 20 == 0 && gImageEffectSuite->abort(instance)) break;

        // get the row start for the output image
        T *dstPix = output.pixelAddress<T>(renderWindow.x1, y);

        for(int x = renderWindow.x1; x < renderWindow.x2; x++) {

            // get the source pixel
            T *srcPix = src.pixelAddress<T>(x, y);

            // get the amount to mask by, no mask image means we do the full effect everywhere
            float maskAmount = 1.0f;
            if (mask) {
                // get our mask pixel address
                T *maskPix = mask.pixelAddress<T>(x, y);
                if(maskPix) {
                    maskAmount = float(*maskPix)/float(MAX);
                }
                else {
                    maskAmount = 0;
                }
            }

            if(srcPix) {
                if(maskAmount == 0) {
                    // we have a mask input, but the mask is zero here,

```

(continues on next page)


```

Image(OfxPropertySetHandle propSet);

// construct from a clip by fetching an image at the given frame
Image(OfxImageClipHandle clip, double frame);

// destructor
~Image();

// get a pixel address, cast to the right type
template <class T>
T *pixelAddress(int x, int y)
{
    return reinterpret_cast<T *>(rawAddress(x, y));
}

// Is this image empty?
operator bool()
{
    return propSet_ != NULL && dataPtr_ != NULL;
}

// bytes per component, 1, 2 or 4 for byte, short and float images
int bytesPerComponent() const { return bytesPerComponent_; }

// number of components
int nComponents() const { return nComponents_; }

protected :
    void construct();

// Look up a pixel address in the image. returns null if the pixel was not
// in the bounds of the image
void *rawAddress(int x, int y);

OfxPropertySetHandle propSet_;
int rowBytes_;
OfxRectI bounds_;
char *dataPtr_;
int nComponents_;
int bytesPerComponent_;
int bytesPerPixel_;
};

```

It takes an `OfxPropertySetHandle` and pulls all the bits it needs out of that into a class. It uses all the same pixel access logic as in example 2. Ideally I should put this in a library which our example links to, but I'm keeping all the code for each example in one source file for illustrative purposes. Feel free to steal this and use it in your own code¹.

¹ provided you stick to the conditions listed at the top of source file

2.32 Summary

This plugin has shown you - the basics of working with multiple contexts, - how to handle optional input clips, - restricting pixel types on input and output clips.

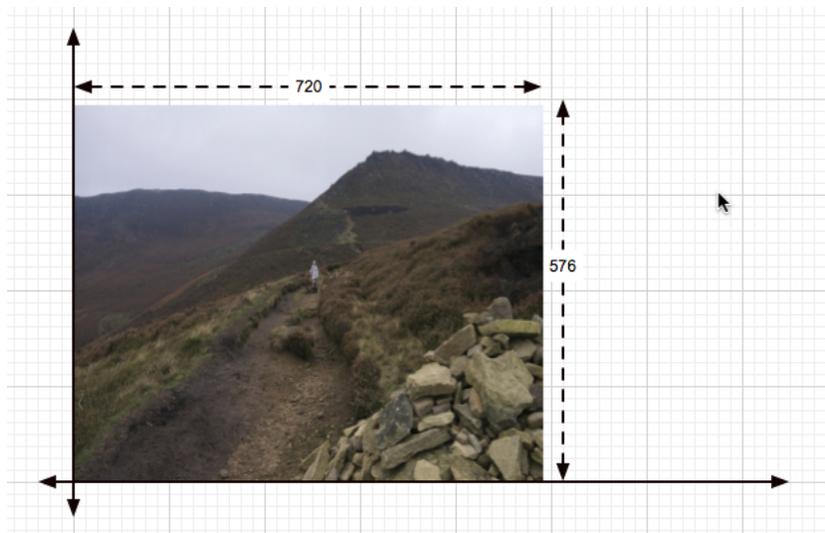
This guide will introduce the spatial coordinate system used by OFX and will illustrate that with a simple circle drawing plugin. Its source can be found in the source code file `circle.cpp`. This plugin takes a clip and draws a circle over it. The colour, size and position of the circle are controlled by several parameters.

2.33 Spatial Coordinate Systems

There are two main coordinate systems in OFX, these are the *pixel coordinates* system and the *canonical coordinates* system. I'll describe them both, but first a slight digression.

2.33.1 Pixel Aspect Ratios

Some display formats (for example standard definition PAL and NTSC) have non square pixels, which is quite annoying in my opinion. The *pixel aspect ratio* defines how non-square your pixel is.



For example, a digital PAL 16:9 wide-screen image has 720 by 576 actual addressable pixels, however it has a pixel aspect ratio of ~ 1.42 ¹.

This means on the display device, each of those pixels are stretched horizontally by a factor of ~ 1.42 . If it were square pixels, our displayed image would actually have 1024 pixels.

Looking at the two images above you can distinctly see the affect that the PAR has, the image appears *squashed* when viewed as raw pixels, but these stretch out to look correct on our final display device.

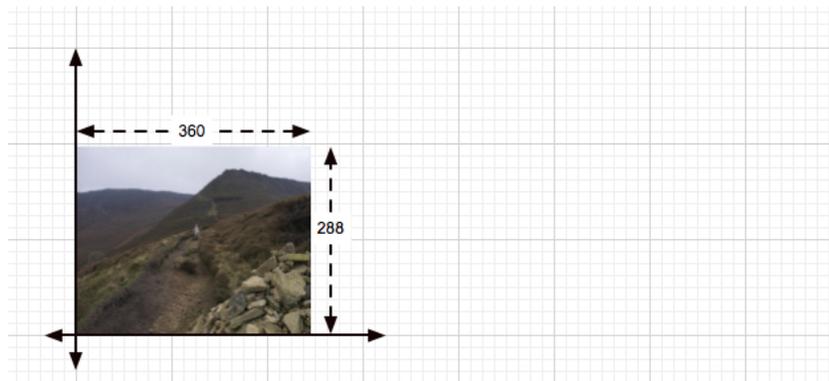
¹ Yes, it can also be 1.46, depending on who you talk to. Today I'm picking 1.42 to force an exact 16 by 9 aspect on a PAL's 720x576 pixels



2.33.2 Render Scales

Applications typically let a user generate low resolution proxy previews of their work, either to save time or space until they have finished. In OFX we call this applying a **render scale**, which has two values one for X and one for Y.

If we were doing a half resolution proxy render of our PAL 16:9 project, we'd have a renderscale of (0.5,0.5), and 360 by 288 addressable pixels in an image, with a PAR of 1.42.



2.33.3 Coordinate Systems

We call the coordinate system which allows us to index honest to goodness pixels in memory, the *pixel coordinates* system. It is usually represented with integers. For example the **bounds** of our image data are in pixel coordinates.

Now expressing positions or sizes on the image plane in pixel coordinates is problematic, both because of pixel aspect ratios and render scales. The problem with sizes is that a circle drawn with a constant radius in pixel coordinates will not necessarily be circular on the final display device, it will be stretched by the pixel aspect ratio. Circles won't be round.

A similar problem applies to render scales, when you say something is at a position, it has to be independent of the renderscale.

To get around this, we have the *canonical coordinates* system. This is on an idealised image plane, where all pixels are square and we have no scales applied to any data. Canonical coordinates are typically represented by doubles.

Back to our PAL 16:9 example. The canonical coordinate system for that image would always have $x=0$ at the left and $x=1023$ at the right, circles are always appear to be round and the arithmetic is easy. We use the canonical coordinate system to express render scale and PAR invariant data. This is the coordinate system we express spatial parameters in.

There was a third coordinate system, the *Normalised Coordinates* System, but in practice this proved to be problematic and has been deprecated.

2.33.4 Mapping Between Coordinate Systems

Obviously on render you will need to map parameters from *canonical coordinates* to the required *pixel coordinates*, or vice versa. That is fortunately very easy, you just need to do a bit of multiplying via the pixel aspect ratio and the renderscale. *See reference for more details.*

2.34 Loading Our Plugin

This plugin highlights the fact that the OFX API is really a way a plugin and a host can have a discussion so they can both figure out how they should operate. It allows plugins to modify their behaviour depending on what the host says it can do.

Here is the source for the load action...

circle.cpp

```

////////////////////////////////////
// The first _action_ called after the binary is loaded (three boot strapper functions.
↳will be however)
OfxStatus LoadAction(void)
{
    // fetch our three suites
    FetchSuite(gPropertySuite,    kOfxPropertySuite,    1);
    FetchSuite(gImageEffectSuite, kOfxImageEffectSuite, 1);
    FetchSuite(gParameterSuite,   kOfxParameterSuite,   1);

    int verSize = 0;
    if(gPropertySuite->propGetDimension(gHost->host, kOfxPropAPIVersion, &verSize) ==
↳kOfxStatOK) {
        verSize = verSize > 2 ? 2 : verSize;
        gPropertySuite->propGetIntN(gHost->host,
                                   kOfxPropAPIVersion,
                                   2,
                                   gAPIVersion);
    }

    // we only support 1.2 and above
    if(gAPIVersion[0] == 1 && gAPIVersion[1] < 2) {
        return kOfxStatFailed;
    }

    /// does the host support multi-resolution images
    gPropertySuite->propGetInt(gHost->host,
                              kOfxImageEffectPropSupportsMultiResolution,
                              0,

```

(continues on next page)

(continued from previous page)

```

        &gHostSupportsMultiRes);

    return kOfxStatOK;
}

```

It fetches three suites then it checks to see if the *kOfxPropAPIVersion* property exists on the host, if it does it then checks that the version is at least “1.2”, as we later rely on features only available in that version of the API.

The next thing it does is to check that the host supports multiple resolutions. This is short hand for saying that the host allows input and output clips to have different regions of definition, and images may be passed to the plugin that have differing bounds. This is also a property of the plugin descriptor, but we’ve left it at the default value, which is *true*, as our plugin does support multiple resolutions.

We are checking for multiple resolution support to conditionally modify our plugin’s behaviour in later actions.

2.35 Description

Now, onto our plugin. The description action is pretty standard, as is the describe in context action. I’ll just show you snippets of the interesting bits.

Note, we are relying on a parameter type that is only available with the 1.2 version of OFX. Our plugin checks for this version of the API the host supports and will fail gracefully during the load action.

circle.cpp

```

// set the properties on the radius param
gParameterSuite->paramDefine(paramSet,
                             kOfxParamTypeDouble,
                             RADIUS_PARAM_NAME,
                             &radiusParamProps);

gPropertySuite->propSetString(radiusParamProps,
                              kOfxParamPropDoubleType,
                              0,
                              kOfxParamDoubleTypeX);

gPropertySuite->propSetString(radiusParamProps,
                              kOfxParamPropDefaultCoordinateSystem,
                              0,
                              kOfxParamCoordinatesNormalised);

gPropertySuite->propSetDouble(radiusParamProps,
                              kOfxParamPropDefault,
                              0,
                              0.25);

gPropertySuite->propSetDouble(radiusParamProps,
                              kOfxParamPropMin,
                              0,
                              0);

gPropertySuite->propSetDouble(radiusParamProps,
                              kOfxParamPropDisplayMin,
                              0,

```

(continues on next page)

(continued from previous page)

```

                                0.0);
gPropertySuite->propSetDouble(radiusParamProps,
                                kOfxParamPropDisplayMax,
                                0,
                                2.0);
gPropertySuite->propSetString(radiusParamProps,
                                kOfxPropLabel,
                                0,
                                "Radius");
gPropertySuite->propSetString(radiusParamProps,
                                kOfxParamPropHint,
                                0,
                                "The radius of the circle.");

```

Here we are defining the parameter that controls the radius of our circle we will draw. It's a double param, and the type of double param is *kOfxParamDoubleTypeX*,² which says to the host, this represents a size in X in canonical coordinates. The host can display that however it like, but to the API, it needs to pass values back in canonical coordinates.

The other thing we do is to set up the default value. Which is 0.25, which seems to be a mighty small circle, as is the display maximum value of 2.0. However, note the property *kOfxParamPropDefaultCoordinateSystem* being set to *kOfxParamCoordinatesNormalised*, this says that defaults/mins/maxes are being described relative to the project size. So our circle's radius will default to be a quarter of the nominal project size's x dimension. For a 1080 HD project, this would be a value of 480.

circle.cpp

```

// set the properties on the centre param
OfxPropertySetHandle centreParamProps;
static double centreDefault[] = {0.5, 0.5};

gParameterSuite->paramDefine(paramSet,
                                kOfxParamTypeDouble2D,
                                CENTRE_PARAM_NAME,
                                &centreParamProps);

gPropertySuite->propSetString(centreParamProps,
                                kOfxParamPropDoubleType,
                                0,
                                kOfxParamDoubleTypeXYAbsolute);
gPropertySuite->propSetString(centreParamProps,
                                kOfxParamPropDefaultCoordinateSystem,
                                0,
                                kOfxParamCoordinatesNormalised);
gPropertySuite->propSetDoubleN(centreParamProps,
                                kOfxParamPropDefault,
                                2,
                                centreDefault);
gPropertySuite->propSetString(centreParamProps,
                                kOfxPropLabel,
                                0,
                                "Centre");
gPropertySuite->propSetString(centreParamProps,

```

(continues on next page)

² this double parameter type is only available API versions 1.2 or above

(continued from previous page)

```

        kOfxParamPropHint,
        0,
        "The centre of the circle.");

```

Here we are defining the parameter that controls the position of the centre of our circle. It's a 2D double parameter and we are telling the host that it represents an absolute position in the canonical coordinate system³. Some hosts will automatically add user interface handles for such parameters to let you simply drag such positions around. We are also setting the default values relative to the project size, and in this case (0.5, 0.5), it should appear in the centre of the final image.

circle.cpp

```

// set the properties on the colour param
OfxPropertySetHandle colourParamProps;
static double colourDefault[] = {1.0, 1.0, 1.0, 0.5};

gParameterSuite->paramDefine(paramSet,
                             kOfxParamTypeRGBA,
                             COLOUR_PARAM_NAME,
                             &colourParamProps);
gPropertySuite->propSetDoubleN(colourParamProps,
                               kOfxParamPropDefault,
                               4,
                               colourDefault);
gPropertySuite->propSetString(colourParamProps,
                              kOfxPropLabel,
                              0,
                              "Colour");
gPropertySuite->propSetString(centreParamProps,
                              kOfxParamPropHint,
                              0,
                              "The colour of the circle.");

```

This is obvious, we are defining an RGBA parameter to control the colour and transparency of our circle. Colours are always normalised 0 to 1, so when you get and set the colour, you need to scale the values up to the nominal white point of your image, which is implicitly defined by the data type of the image.

circle.cpp

```

if(gHostSupportsMultiRes) {
    OfxPropertySetHandle growRodParamProps;
    gParameterSuite->paramDefine(paramSet,
                                 kOfxParamTypeBoolean,
                                 GROW_ROD_PARAM_NAME,
                                 &growRodParamProps);
    gPropertySuite->propSetInt(growRodParamProps,
                              kOfxParamPropDefault,
                              0,
                              0);
    gPropertySuite->propSetString(growRodParamProps,
                                 kOfxParamPropHint,
                                 0,

```

(continues on next page)

³ this double parameter type is only available API versions 1.2 or above

(continued from previous page)

```

                                "Whether to grow the output's Region of Definition to
↪include the circle.");
    gPropertySuite->propSetString(growRoDParamProps,
                                kOfxPropLabel,
                                0,
                                "Grow RoD");
}

```

Finally, we are conditionally defining a boolean parameter that controls whether our circle affects the region of definition of our output image. We only able to modify the region of definition if the host has an architecture that supports that behaviour, which we checked at load time where we set the **gHostSupportsMultiRes** global variable.

2.36 Get Region Of Definition Action

What is this region of definition action? Easy, an effect and a clip have a region of definition (RoD). This is the maximum rectangle for which an effect or clip can produce pixels. You can ask for RoD of a clip via the *OfxImageEffectSuiteV1::clipGetRegionOfDefinition()* function in the image effect suite. The RoD is currently defined in canonical coordinates⁴.

Note that the RoD is independent of the **bounds** of a image, an image's bounds may be less than, more than or equal to the RoD. It is up to host how or why it wants to manage the RoD differently. As noted above, some hosts don't have the ability to do any such thing.

By default the RoD of the output is the union of all the RoDs from all the mandatory input clips. In our example, we want to be able to set the RoD to be the union of the input clip with the area the circle we are drawing. Whether we do that or not is controlled by the "growRoD" parameter which is conditionally defined in the describe in context action.

To set the output rod, we need to trap the *kOfxImageEffectActionGetRegionOfDefinition* action. Our MainEntry function now has an extra conditional in there...

circle.cpp

```

...
else if(gHostSupportsMultiRes && strcmp(action,
↪kOfxImageEffectActionGetRegionOfDefinition) == 0) {
    returnStatus = GetRegionOfDefinitionAction(effect, inArgs, outArgs);
}
...

```

Note that we don't trap this on hosts that aren't multi-resolution, as by definition on those hosts RoDs are fixed.

The code for the action itself is quite simple:

circle.cpp

```

// tells the host what region we are capable of filling
OfxStatus
GetRegionOfDefinitionAction( OfxImageEffectHandle effect,
                             OfxPropertySetHandle inArgs,
                             OfxPropertySetHandle outArgs)
{
    // retrieve any instance data associated with this effect

```

(continues on next page)

⁴ we are debating whether to modifying that to be in pixel coordinates

```

MyInstanceData *myData = FetchInstanceData(effect);

OfxTime time;
gPropertySuite->propGetDouble(inArgs, kOfxPropTime, 0, &time);

int growingRoD;
gParameterSuite->paramGetValueAtTime(myData->growRoD, time,
                                     &growingRoD);

// are we growing the RoD to include the circle?
if(not growingRoD) {
    return kOfxStatReplyDefault;
}
else {
    double radius = 0.0;
    gParameterSuite->paramGetValueAtTime(myData->radiusParam, time,
                                         &radius);

    double centre[2];
    gParameterSuite->paramGetValueAtTime(myData->centreParam, time,
                                         &centre[0],
                                         &centre[1]);

    // get the source rod
    OfxRectD rod;
    gImageEffectSuite->clipGetRegionOfDefinition(myData->sourceClip, time, &rod);

    if(rod.x1 > centre[0] - radius) rod.x1 = centre[0] - radius;
    if(rod.y1 > centre[1] - radius) rod.y1 = centre[1] - radius;

    if(rod.x2 < centre[0] + radius) rod.x2 = centre[0] + radius;
    if(rod.y2 < centre[1] + radius) rod.y2 = centre[1] + radius;

    // set the rod in the out args
    gPropertySuite->propSetDoubleN(outArgs, kOfxImageEffectPropRegionOfDefinition, 4, &
    ↪rod.x1);

    // and say we trapped the action and we are at the identity
    return kOfxStatOK;
}
}

```

We are being asked to calculate the RoD at a specific time, which means that RoDs are time varying in OFX.

We check our *growRoD* parameter to see if we are going to actually modify the RoD. If we do, we find out, in canonical coordinates, where we are drawing our circle. We then fetch the region of definition and make a union of those two regions. We then set the *kOfxImageEffectPropRegionOfDefinition* return property on **outArgs** and say that we trapped the action.

All fairly easy.

2.37 Is Identity Action

Our identity checking action is fairly obvious, we check to see if our circle has a non zero radius, and to see if we are not growing the RoD and our circle is outside the RoD.

2.38 Rendering

The action code is fairly boiler plate, it fetches parameter values and images from clips before calling the templated PixelProcessing function. Which is below:

circle.cpp

```
template <class T, int MAX>
void PixelProcessing(OfxImageEffectHandle instance,
                   Image &src,
                   Image &output,
                   double centre[2],
                   double radius,
                   double colour[4],
                   double renderScale[2],
                   OfxRectI renderWindow)
{
    // pixel aspect of our output
    float PAR = output.pixelAspectRatio();

    T colourQuantised[4];
    for(int c = 0; c < 4; ++c) {
        colourQuantised[c] = Clamp<T, MAX>(colour[c] * MAX);
    }

    // now do some processing
    for(int y = renderWindow.y1; y < renderWindow.y2; y++) {
        if(y % 20 == 0 && gImageEffectSuite->abort(instance)) break;

        // get our y coord in canonical space
        float yCanonical = (y + 0.5f)/renderScale[1];

        // how far are we from the centre in y, canonical
        float dy = yCanonical - centre[1];

        // get the row start for the output image
        T *dstPix = output.pixelAddress<T>(renderWindow.x1, y);

        for(int x = renderWindow.x1; x < renderWindow.x2; x++) {
            // get our x pixel coord in canonical space,
            float xCanonical = (x + 0.5) * PAR/renderScale[0];

            // how far are we from the centre in x, canonical
            float dx = xCanonical - centre[0];

            // distance to the centre of our circle, canonical
            float d = sqrtf(dx * dx + dy * dy);
```

(continues on next page)

(continued from previous page)

```
// this will hold the antialiased value
float alpha = colour[3];

// Is the square of the distance to the centre
// less than the square of the radius?
if(d < radius) {
    if(d > radius - 1) {
        // we are within 1 pixel of the edge, modulate
        // our alpha with an anti-aliasing value
        alpha *= radius - d;
    }
}
else {
    // outside, so alpha is 0
    alpha = 0;
}

// get the source pixel
const T *srcPix = src.pixelAddressWithFallback<T>(x, y);

// scale each component around that average
for(int c = 0; c < output.nComponents(); ++c) {
    // use the mask to control how much original we should have
    dstPix[c] = Blend(srcPix[c], colourQuantised[c], alpha);
}
dstPix += output.nComponents();
}
}
```

Please don't think I actually write production code as slow as this, I'm just making the whole thing as clear as possible in my example.

The first thing we do is to scale the normalised value for our circle colour up to a quantised value based on our data type. So multiplying up by 255 for 8 bit data types, 65536 for 16bit ints and 1 for floats.

To draw the circle we are transforming a pixel's position in pixel space into a canonical coordinate. We then calculate the distance to the centre of the circle, again in canonical coordinates. We use that distance to see if we are inside or out of the circle, with a bit of anti-aliasing thrown in. This gives us a normalised alpha value.

Our output value is our source pixel blended with our circle colour based on the intensity of the calculated alpha.

2.39 Summary

This example plugin has shown ...

- the two main OFX spatial coordinate systems,
- how to use the region of definition action,
- that the API is a negotiation between a host and a plugin,
- mapping between coordinate systems for rendering.

This directory tree contains a set of plugins and corresponding guides which take you through the basics of the OFX Image Effects Plugin API.

There are two sub-directories...

- Code - which contains the example plugins source files,
- Doc - has a guide to each plugin.

+++++
BUILDING THE PLUGINS

The plugins and support libs are set up to use cmake to generate and run builds. See the top-level README.md in the repo, and use `scripts/build-cmake.sh`.

+++++
BUILDING THE DOCUMENTATION

See the file Documentation/README, and use `Documentation/build.sh`. The docs are also auto-generated on commit to the main branch on github.

This **is** a brief description of examples which could do **with** being added to the Guides.

The following examples should be added to the guide.

OpenGL Overlay Example

=====

A trivial example along the lines of the circle drawing example.

This will illustrate...

- openGL overlays

Temporal Difference Example

=====

Compute the absolute difference between images at two different times.

This will illustrate...

- fetching images at separate times
- the get frames needed action

Transition Example

=====

A simple straight frame blend transition

This will illustrate...

- the transition context

Custom Parameter

=====

Something like the circle drawing plugin so that it has no double parameters, but only a a.

(continues on next page)

↪ single custom parameter, controlled via the overlay UI.

This will illustrate...

- using custom parameters.

3x3 2D Filter

=====

Does a variety of simple 2D filtering operations using a 3x3 window.

This will illustrate...

- the get region of interest action

Analysis Plugin

=====

The plugin will find the minimum **and** maximum value of a given frame **in** response to a ↪ push button, **and** store the values into two parameters. During renders it will rescale pixel values so that **'min' is 0 and** ↪ **'max' is** the whitepoint.

This will illustrate...

- writing to parameters **in** response to a button press outside of render.

Basic OpenGL Example

=====

Behaviour to be determined.

This will illustrate the basics of rendering **in** OpenGL

Last Edit 11/11/14

OPENFX RELEASE NOTES

3.1 OpenFX Release Notes for V1.4

Release Date: Sept 1, 2015

This is version 1.4 of the OpenFX API. Significant additions include a Dialog Suite for plugins to request the host to allow them to put up a modal dialog, a NativeOrigin host property, draft render quality support, half-float format tag available for CPU rendering (not just OpenGL), and a new internationalizable version of the progress suite.

A number of ambiguities in the spec have also been clarified in this version, including allowing OpenGL processing and tiled rendering to be enabled/disabled in Instance Changed events, and clarifying the semantics of dialogs and the progress suite. The old Analysis pass action has also been removed.

3.1.1 New Suites

- `OfxDialogSuiteV1`

3.1.2 New Suite Versions

- `OfxProgressSuiteV2`

3.1.3 New Properties

- `kOfxImageEffectPropRenderQualityDraft`
- `kOfxImageEffectHostPropNativeOrigin`

3.1.4 Deprecations

None

3.1.5 Removals

- Analysis pass action
- `kOfxImageComponentYUVA` and related: YUVA pixel formats
- `kOfxInteractPropViewportSize`
- `kOfxParamPropPluginMayWrite`
- `kOfxImageEffectPropInAnalysis`
- `kOfxParamDoubleTypeNormalised*` - removed in favor of `kOfxParamDoubleType*`

3.2 OpenFX Release Notes for V1.5

Release Date: TBD

This will be version 1.5 of the OpenFX API. Significant additions support for Metal, CUDA and OpenCL rendering (including half-float bit depths), and an `OfxDrawSuite` for drawing overlays without requiring OpenGL. It also includes a new `PropChoiceValue` string-valued param type and the ability to reorder `PropChoice` options in new versions of a plugin while retaining back compatibility.

This is the first release produced by new OpenFX project within the [Academy Software Foundation](#).

The build process is updated to use CMake and Conan, docs are available now on [ReadTheDocs](#), and we have a new website at <https://openeffects.org>. Builds of the headers and support libs and plugins are now automatically produced on all changes to *main* on our [Github](#).

3.2.1 GPU Rendering

The GPU Rendering Suite now supports Metal, CUDA, and OpenCL (Images and Buffers). See [Rendering on GPU](#) and source file `include/ofxGPURender.h` for details.

3.2.2 New Suites

- `OfxDrawSuiteV1`

3.2.3 New Suite Versions

None

3.2.4 New Actions

- `OfxSetHost()` – called first by the host, if it exists

3.2.5 New Properties

String Choice params:

- *kOfxParamTypeStrChoice* – string-valued choice (menu) parameter
- *kOfxParamPropChoiceOrder* – specifies the order in which ChoiceParam options are presented

GPU rendering:

- *kOfxImageEffectPropMetalRenderSupported*
 - *kOfxImageEffectPropCudaRenderSupported*
 - *kOfxImageEffectPropOpenCLRenderSupported*
 - *kOfxImageEffectPropMetalEnabled*
 - *kOfxImageEffectPropCudaEnabled*
 - *kOfxImageEffectPropOpenCLEnabled*
- etc.

3.2.6 Deprecations

None

3.2.7 Removals

None

Symbols

__OFXGPURENDER_H__ (C macro), 231, 476
 _ofxImageEffect_h_ (C macro), 242, 484
 _ofxOpenGLRender_h_ (C macro), 316, 545

K

kOfxActionBeginInstanceChanged (C macro), 83, 211, 462
 kOfxActionBeginInstanceEdit (C macro), 86, 212, 463
 kOfxActionCreateInstance (C macro), 82, 209, 460
 kOfxActionCreateInstanceInteract (C macro), 95, 279, 515
 kOfxActionDescribe (C macro), 81, 207, 457
 kOfxActionDescribeInteract (C macro), 95, 279, 515
 kOfxActionDestroyInstance (C macro), 82, 209, 460
 kOfxActionDestroyInstanceInteract (C macro), 96, 279, 516
 kOfxActionDialog (C macro), 225, 472
 kOfxActionEndInstanceChanged (C macro), 84, 212, 463
 kOfxActionEndInstanceEdit (C macro), 86, 212, 463
 kOfxActionInstanceChanged (C macro), 84, 210, 461
 kOfxActionLoad (C macro), 80, 206, 457
 kOfxActionOpenGLContextAttached (C macro), 233, 478, 572
 kOfxActionOpenGLContextDetached (C macro), 234, 479, 573
 kOfxActionPurgeCaches (C macro), 85, 208, 459
 kOfxActionSyncPrivateData (C macro), 85, 208, 459
 kOfxActionUnload (C macro), 81, 207, 458
 kOfxBitDepthByte (C macro), 77, 217, 469
 kOfxBitDepthFloat (C macro), 78, 218, 469
 kOfxBitDepthHalf (C macro), 78, 218, 469
 kOfxBitDepthNone (C macro), 78, 217, 468
 kOfxBitDepthShort (C macro), 77, 218, 469
 kOfxChangePluginEdited (C macro), 50, 217, 468
 kOfxChangeTime (C macro), 50, 217, 468
 kOfxChangeUserEdited (C macro), 50, 217, 468
 kOfxDialogSuite (C macro), 225, 472
 kOfxDrawSuite (C macro), 226, 473
 kOfxFlagInfiniteMax (C macro), 22, 217, 468
 kOfxFlagInfiniteMin (C macro), 22, 217, 468
 kOfxHostNativeOriginBottomLeft (C macro), 255, 497
 kOfxHostNativeOriginCenter (C macro), 255, 497
 kOfxHostNativeOriginTopLeft (C macro), 255, 497
 kOfxImageClipPropConnected (C macro), 159, 263, 505
 kOfxImageClipPropContinuousSamples (C macro), 159, 259, 501
 kOfxImageClipPropFieldExtraction (C macro), 159, 268, 510
 kOfxImageClipPropFieldOrder (C macro), 160, 266, 508
 kOfxImageClipPropIsMask (C macro), 160, 261, 503
 kOfxImageClipPropOptional (C macro), 160, 261, 503
 kOfxImageClipPropUnmappedComponents (C macro), 160, 260, 502
 kOfxImageClipPropUnmappedPixelDepth (C macro), 161, 259, 501
 kOfxImageComponentAlpha (C macro), 78, 242, 484
 kOfxImageComponentNone (C macro), 78, 242, 484
 kOfxImageComponentRGB (C macro), 78, 242, 484
 kOfxImageComponentRGBA (C macro), 78, 242, 484
 kOfxImageComponentYUVA (C macro), 313, 543
 kOfxImageEffectActionBeginSequenceRender (C macro), 92, 248, 490
 kOfxImageEffectActionDescribeInContext (C macro), 87, 250, 492
 kOfxImageEffectActionEndSequenceRender (C macro), 92, 249, 491
 kOfxImageEffectActionGetClipPreferences (C macro), 93, 246, 488
 kOfxImageEffectActionGetFramesNeeded (C macro), 89, 245, 487
 kOfxImageEffectActionGetRegionOfDefinition (C macro), 87, 243, 485
 kOfxImageEffectActionGetRegionsOfInterest (C macro), 88, 244, 486
 kOfxImageEffectActionGetTimeDomain (C macro), 94, 244, 486

- `kOfxImageEffectActionIsIdentity` (*C macro*), 90, 247, 489
- `kOfxImageEffectActionRender` (*C macro*), 91, 248, 489
- `kOfxImageEffectContextFilter` (*C macro*), 242, 484
- `kOfxImageEffectContextGeneral` (*C macro*), 242, 484
- `kOfxImageEffectContextGenerator` (*C macro*), 242, 484
- `kOfxImageEffectContextPaint` (*C macro*), 242, 484
- `kOfxImageEffectContextRetimer` (*C macro*), 242, 484
- `kOfxImageEffectContextTransition` (*C macro*), 242, 484
- `kOfxImageEffectFrameVarying` (*C macro*), 161, 263, 505
- `kOfxImageEffectHostPropIsBackground` (*C macro*), 161, 251, 493
- `kOfxImageEffectHostPropNativeOrigin` (*C macro*), 255, 497
- `kOfxImageEffectInstancePropEffectDuration` (*C macro*), 161, 266, 508
- `kOfxImageEffectInstancePropSequentialRender` (*C macro*), 162, 254, 496
- `kOfxImageEffectOutputClipName` (*C macro*), 270, 512
- `kOfxImageEffectPluginApi` (*C macro*), 15, 242, 484
- `kOfxImageEffectPluginApiVersion` (*C macro*), 242, 484
- `kOfxImageEffectPluginPropFieldRenderTwiceAlways` (*C macro*), 64, 162, 268, 510
- `kOfxImageEffectPluginPropGrouping` (*C macro*), 163, 256, 498
- `kOfxImageEffectPluginPropHostFrameThreading` (*C macro*), 163, 252, 494
- `kOfxImageEffectPluginPropOverlayInteractV1` (*C macro*), 163, 256, 498
- `kOfxImageEffectPluginPropOverlayInteractV2` (*C macro*), 163, 257, 498
- `kOfxImageEffectPluginPropSingleInstance` (*C macro*), 164, 251, 493
- `kOfxImageEffectPluginRenderThreadSafety` (*C macro*), 164, 252, 494
- `kOfxImageEffectPropClipPreferencesSlaveParam` (*C macro*), 164, 253, 495
- `kOfxImageEffectPropComponents` (*C macro*), 164, 258, 500
- `kOfxImageEffectPropContext` (*C macro*), 165, 258, 500
- `kOfxImageEffectPropCudaEnabled` (*C macro*), 68, 142, 165, 235, 479, 574
- `kOfxImageEffectPropCudaRenderSupported` (*C macro*), 68, 142, 166, 234, 479, 574
- `kOfxImageEffectPropCudaStream` (*C macro*), 69, 143, 166, 235, 480, 574
- `kOfxImageEffectPropCudaStreamSupported` (*C macro*), 69, 142, 166, 235, 480, 574
- `kOfxImageEffectPropFieldToRender` (*C macro*), 167, 268, 510
- `kOfxImageEffectPropFrameRange` (*C macro*), 167, 263, 505
- `kOfxImageEffectPropFrameRate` (*C macro*), 167, 262, 504
- `kOfxImageEffectPropFrameStep` (*C macro*), 167, 262, 504
- `kOfxImageEffectPropInAnalysis` (*C macro*), 168, 313, 543
- `kOfxImageEffectPropInteractiveRenderStatus` (*C macro*), 168, 256, 497
- `kOfxImageEffectPropMetalCommandQueue` (*C macro*), 70, 144, 168, 237, 481, 576
- `kOfxImageEffectPropMetalEnabled` (*C macro*), 70, 144, 169, 236, 481, 575
- `kOfxImageEffectPropMetalRenderSupported` (*C macro*), 70, 143, 169, 236, 480, 575
- `kOfxImageEffectPropOpenCLCommandQueue` (*C macro*), 72, 146, 169, 238, 482, 577
- `kOfxImageEffectPropOpenCLEnabled` (*C macro*), 72, 145, 170, 237, 482, 577
- `kOfxImageEffectPropOpenCLImage` (*C macro*), 72, 146, 238, 483, 578
- `kOfxImageEffectPropOpenCLRenderSupported` (*C macro*), 71, 145, 170, 237, 481, 576
- `kOfxImageEffectPropOpenCLSupported` (*C macro*), 71, 145, 237, 482, 577
- `kOfxImageEffectPropOpenGLEnabled` (*C macro*), 170, 232, 477, 571
- `kOfxImageEffectPropOpenGLRenderSupported` (*C macro*), 171, 231, 476, 570
- `kOfxImageEffectPropOpenGLTextureIndex` (*C macro*), 171, 233, 477, 571
- `kOfxImageEffectPropOpenGLTextureTarget` (*C macro*), 172, 233, 478, 572
- `kOfxImageEffectPropPixelDepth` (*C macro*), 172, 258, 500
- `kOfxImageEffectPropPluginHandle` (*C macro*), 172, 251, 493
- `kOfxImageEffectPropPreMultiplication` (*C macro*), 173, 260, 502
- `kOfxImageEffectPropProjectExtent` (*C macro*), 173, 265, 507
- `kOfxImageEffectPropProjectOffset` (*C macro*), 173, 265, 507
- `kOfxImageEffectPropProjectPixelAspectRatio` (*C macro*), 173, 266, 508
- `kOfxImageEffectPropProjectSize` (*C macro*), 174, 265, 507
- `kOfxImageEffectPropRegionOfDefinition` (*C*

macro), 174, 269, 511
kOfxImageEffectPropRegionOfInterest (*C macro*), 174, 269, 511
kOfxImageEffectPropRenderQualityDraft (*C macro*), 174, 264, 506
kOfxImageEffectPropRenderScale (*C macro*), 175, 264, 506
kOfxImageEffectPropRenderWindow (*C macro*), 175, 269, 511
kOfxImageEffectPropSequentialRenderStatus (*C macro*), 175, 255, 497
kOfxImageEffectPropSetableFielding (*C macro*), 176, 254, 496
kOfxImageEffectPropSetableFrameRate (*C macro*), 176, 254, 495
kOfxImageEffectPropSupportedComponents (*C macro*), 176, 261, 503
kOfxImageEffectPropSupportedContexts (*C macro*), 177, 251, 492
kOfxImageEffectPropSupportedPixelDepths (*C macro*), 177, 261, 503
kOfxImageEffectPropSupportsMultipleClipDepths (*C macro*), 178, 253, 494
kOfxImageEffectPropSupportsMultipleClipPARs (*C macro*), 178, 253, 495
kOfxImageEffectPropSupportsMultiResolution (*C macro*), 178, 257, 499
kOfxImageEffectPropSupportsOverlays (*C macro*), 179, 256, 498
kOfxImageEffectPropSupportsTiles (*C macro*), 179, 257, 499
kOfxImageEffectPropTemporalClipAccess (*C macro*), 179, 258, 499
kOfxImageEffectPropUnmappedFrameRange (*C macro*), 179, 263, 505
kOfxImageEffectPropUnmappedFrameRate (*C macro*), 180, 262, 504
kOfxImageEffectRenderFullySafe (*C macro*), 61, 252, 494
kOfxImageEffectRenderInstanceSafe (*C macro*), 61, 252, 494
kOfxImageEffectRenderUnsafe (*C macro*), 61, 252, 494
kOfxImageEffectRetimerParamName (*C macro*), 270, 512
kOfxImageEffectSimpleSourceClipName (*C macro*), 270, 512
kOfxImageEffectSuite (*C macro*), 270, 512
kOfxImageEffectTransitionParamName (*C macro*), 270, 512
kOfxImageEffectTransitionSourceFromClipName (*C macro*), 270, 512
kOfxImageEffectTransitionSourceToClipName (*C macro*), 270, 512
kOfxImageFieldBoth (*C macro*), 63, 270, 512
kOfxImageFieldDoubled (*C macro*), 270, 512
kOfxImageFieldLower (*C macro*), 63, 64, 269, 511
kOfxImageFieldNone (*C macro*), 63, 269, 511
kOfxImageFieldSingle (*C macro*), 270, 512
kOfxImageFieldUpper (*C macro*), 63, 64, 270, 512
kOfxImageOpaque (*C macro*), 38, 260, 502
kOfxImagePreMultiplied (*C macro*), 38, 260, 502
kOfxImagePropBounds (*C macro*), 180, 266, 508
kOfxImagePropData (*C macro*), 180, 266, 508
kOfxImagePropField (*C macro*), 180, 267, 509
kOfxImagePropPixelAspectRatio (*C macro*), 181, 262, 504
kOfxImagePropRegionOfDefinition (*C macro*), 181, 267, 509
kOfxImagePropRowBytes (*C macro*), 181, 267, 509
kOfxImagePropUniqueIdentifier (*C macro*), 182, 259, 501
kOfxImageUnPreMultiplied (*C macro*), 38, 260, 502
kOfxInteractActionDraw (*C macro*), 96, 280, 516
kOfxInteractActionGainFocus (*C macro*), 101, 285, 521
kOfxInteractActionKeyDown (*C macro*), 99, 283, 519
kOfxInteractActionKeyRepeat (*C macro*), 100, 284, 520
kOfxInteractActionKeyUp (*C macro*), 100, 284, 520
kOfxInteractActionLoseFocus (*C macro*), 101, 285, 522
kOfxInteractActionPenDown (*C macro*), 98, 281, 518
kOfxInteractActionPenMotion (*C macro*), 97, 281, 517
kOfxInteractActionPenUp (*C macro*), 98, 282, 518
kOfxInteractPropBackgroundColour (*C macro*), 182, 277, 513
kOfxInteractPropBitDepth (*C macro*), 182, 278, 515
kOfxInteractPropDrawContext (*C macro*), 149, 182, 226, 473
kOfxInteractPropHasAlpha (*C macro*), 182, 278, 515
kOfxInteractPropPenPosition (*C macro*), 183, 278, 514
kOfxInteractPropPenPressure (*C macro*), 183, 278, 514
kOfxInteractPropPenViewportPosition (*C macro*), 183, 278, 514
kOfxInteractPropPixelScale (*C macro*), 183, 277, 513
kOfxInteractPropSlaveToParam (*C macro*), 183, 277, 513
kOfxInteractPropSuggestedColour (*C macro*), 184, 277, 514
kOfxInteractPropViewportSize (*C macro*), 184, 313, 543
kOfxInteractSuite (*C macro*), 277, 513
kOfxKey_0 (*C macro*), 296, 532

kOfxKey_1 (C macro), 296, 532
kOfxKey_2 (C macro), 296, 532
kOfxKey_3 (C macro), 296, 532
kOfxKey_4 (C macro), 296, 532
kOfxKey_5 (C macro), 296, 532
kOfxKey_6 (C macro), 296, 532
kOfxKey_7 (C macro), 297, 532
kOfxKey_8 (C macro), 297, 533
kOfxKey_9 (C macro), 297, 533
kOfxKey_A (C macro), 297, 533
kOfxKey_a (C macro), 299, 535
kOfxKey_Aacute (C macro), 302, 538
kOfxKey_aacute (C macro), 303, 539
kOfxKey_Acircumflex (C macro), 302, 538
kOfxKey_acircumflex (C macro), 303, 539
kOfxKey_acute (C macro), 301, 537
kOfxKey_Adiaeresis (C macro), 302, 538
kOfxKey_adiaeresis (C macro), 303, 539
kOfxKey_AE (C macro), 302, 538
kOfxKey_ae (C macro), 303, 539
kOfxKey_Agrave (C macro), 302, 537
kOfxKey_agrave (C macro), 303, 539
kOfxKey_Alt_L (C macro), 295, 531
kOfxKey_Alt_R (C macro), 295, 531
kOfxKey_ampersand (C macro), 296, 532
kOfxKey_apostrophe (C macro), 296, 532
kOfxKey_Aring (C macro), 302, 538
kOfxKey_aring (C macro), 303, 539
kOfxKey_asciicircum (C macro), 298, 534
kOfxKey_asciitilde (C macro), 300, 536
kOfxKey_asterisk (C macro), 296, 532
kOfxKey_at (C macro), 297, 533
kOfxKey_Atilde (C macro), 302, 538
kOfxKey_atilde (C macro), 303, 539
kOfxKey_B (C macro), 297, 533
kOfxKey_b (C macro), 299, 535
kOfxKey_backslash (C macro), 298, 534
kOfxKey_BackSpace (C macro), 287, 523
kOfxKey_bar (C macro), 300, 536
kOfxKey_Begin (C macro), 289, 525
kOfxKey_braceleft (C macro), 300, 536
kOfxKey_braceright (C macro), 300, 536
kOfxKey_bracketleft (C macro), 298, 534
kOfxKey_bracketright (C macro), 298, 534
kOfxKey_Break (C macro), 290, 526
kOfxKey_brokenbar (C macro), 300, 536
kOfxKey_C (C macro), 297, 533
kOfxKey_c (C macro), 299, 535
kOfxKey_Cancel (C macro), 290, 526
kOfxKey_Caps_Lock (C macro), 295, 531
kOfxKey_Ccedilla (C macro), 302, 538
kOfxKey_ccedilla (C macro), 303, 539
kOfxKey_cedilla (C macro), 301, 537
kOfxKey_cent (C macro), 300, 536
kOfxKey_Clear (C macro), 287, 523
kOfxKey_colon (C macro), 297, 533
kOfxKey_comma (C macro), 296, 532
kOfxKey_Control_L (C macro), 295, 531
kOfxKey_Control_R (C macro), 295, 531
kOfxKey_copyright (C macro), 300, 536
kOfxKey_currency (C macro), 300, 536
kOfxKey_D (C macro), 297, 533
kOfxKey_d (C macro), 299, 535
kOfxKey_degree (C macro), 301, 537
kOfxKey_Delete (C macro), 288, 524
kOfxKey_diaeresis (C macro), 300, 536
kOfxKey_division (C macro), 304, 540
kOfxKey_dollar (C macro), 296, 532
kOfxKey_Down (C macro), 289, 525
kOfxKey_E (C macro), 297, 533
kOfxKey_e (C macro), 299, 535
kOfxKey_Eacute (C macro), 302, 538
kOfxKey_eacute (C macro), 304, 539
kOfxKey_Ecircumflex (C macro), 302, 538
kOfxKey_ecircumflex (C macro), 304, 540
kOfxKey_Ediaeresis (C macro), 302, 538
kOfxKey_ediaeresis (C macro), 304, 540
kOfxKey_Egrave (C macro), 302, 538
kOfxKey_egrave (C macro), 303, 539
kOfxKey_Eisu_Shift (C macro), 289, 525
kOfxKey_Eisu_toggle (C macro), 289, 525
kOfxKey_End (C macro), 289, 525
kOfxKey_equal (C macro), 297, 533
kOfxKey_Escape (C macro), 288, 524
kOfxKey_ETH (C macro), 302, 538
kOfxKey_Eth (C macro), 302, 538
kOfxKey_eth (C macro), 304, 540
kOfxKey_exclam (C macro), 295, 531
kOfxKey_exclamdown (C macro), 300, 536
kOfxKey_Execute (C macro), 290, 525
kOfxKey_F (C macro), 297, 533
kOfxKey_f (C macro), 299, 535
kOfxKey_F1 (C macro), 292, 528
kOfxKey_F10 (C macro), 292, 528
kOfxKey_F11 (C macro), 292, 528
kOfxKey_F12 (C macro), 292, 528
kOfxKey_F13 (C macro), 293, 528
kOfxKey_F14 (C macro), 293, 529
kOfxKey_F15 (C macro), 293, 529
kOfxKey_F16 (C macro), 293, 529
kOfxKey_F17 (C macro), 293, 529
kOfxKey_F18 (C macro), 293, 529
kOfxKey_F19 (C macro), 293, 529
kOfxKey_F2 (C macro), 292, 528
kOfxKey_F20 (C macro), 293, 529
kOfxKey_F21 (C macro), 293, 529
kOfxKey_F22 (C macro), 293, 529
kOfxKey_F23 (C macro), 293, 529

kOfxKey_F24 (C macro), 294, 530
 kOfxKey_F25 (C macro), 294, 530
 kOfxKey_F26 (C macro), 294, 530
 kOfxKey_F27 (C macro), 294, 530
 kOfxKey_F28 (C macro), 294, 530
 kOfxKey_F29 (C macro), 294, 530
 kOfxKey_F3 (C macro), 292, 528
 kOfxKey_F30 (C macro), 294, 530
 kOfxKey_F31 (C macro), 294, 530
 kOfxKey_F32 (C macro), 294, 530
 kOfxKey_F33 (C macro), 294, 530
 kOfxKey_F34 (C macro), 295, 530
 kOfxKey_F35 (C macro), 295, 531
 kOfxKey_F4 (C macro), 292, 528
 kOfxKey_F5 (C macro), 292, 528
 kOfxKey_F6 (C macro), 292, 528
 kOfxKey_F7 (C macro), 292, 528
 kOfxKey_F8 (C macro), 292, 528
 kOfxKey_F9 (C macro), 292, 528
 kOfxKey_Find (C macro), 290, 526
 kOfxKey_G (C macro), 297, 533
 kOfxKey_g (C macro), 299, 535
 kOfxKey_grave (C macro), 298, 534
 kOfxKey_greater (C macro), 297, 533
 kOfxKey_guillemotleft (C macro), 301, 536
 kOfxKey_guillemotright (C macro), 301, 537
 kOfxKey_H (C macro), 297, 533
 kOfxKey_h (C macro), 299, 535
 kOfxKey_Hankaku (C macro), 288, 524
 kOfxKey_Help (C macro), 290, 526
 kOfxKey_Henkan (C macro), 288, 524
 kOfxKey_Henkan_Mode (C macro), 288, 524
 kOfxKey_Hiragana (C macro), 288, 524
 kOfxKey_Hiragana_Katakana (C macro), 288, 524
 kOfxKey_Home (C macro), 289, 525
 kOfxKey_Hyper_L (C macro), 295, 531
 kOfxKey_Hyper_R (C macro), 295, 531
 kOfxKey_hyphen (C macro), 301, 537
 kOfxKey_I (C macro), 297, 533
 kOfxKey_i (C macro), 299, 535
 kOfxKey_Iacute (C macro), 302, 538
 kOfxKey_iacute (C macro), 304, 540
 kOfxKey_Icircumflex (C macro), 302, 538
 kOfxKey_ircumflex (C macro), 304, 540
 kOfxKey_Idiaeresis (C macro), 302, 538
 kOfxKey_idiaeresis (C macro), 304, 540
 kOfxKey_Igrave (C macro), 302, 538
 kOfxKey_igrave (C macro), 304, 540
 kOfxKey_Insert (C macro), 290, 526
 kOfxKey_J (C macro), 297, 533
 kOfxKey_j (C macro), 299, 535
 kOfxKey_K (C macro), 297, 533
 kOfxKey_k (C macro), 299, 535
 kOfxKey_Kana_Lock (C macro), 289, 525
 kOfxKey_Kana_Shift (C macro), 289, 525
 kOfxKey_Kanji (C macro), 288, 524
 kOfxKey_Katakana (C macro), 288, 524
 kOfxKey_KP_0 (C macro), 291, 527
 kOfxKey_KP_1 (C macro), 291, 527
 kOfxKey_KP_2 (C macro), 291, 527
 kOfxKey_KP_3 (C macro), 292, 527
 kOfxKey_KP_4 (C macro), 292, 528
 kOfxKey_KP_5 (C macro), 292, 528
 kOfxKey_KP_6 (C macro), 292, 528
 kOfxKey_KP_7 (C macro), 292, 528
 kOfxKey_KP_8 (C macro), 292, 528
 kOfxKey_KP_9 (C macro), 292, 528
 kOfxKey_KP_Add (C macro), 291, 527
 kOfxKey_KP_Begin (C macro), 291, 527
 kOfxKey_KP_Decimal (C macro), 291, 527
 kOfxKey_KP_Delete (C macro), 291, 527
 kOfxKey_KP_Divide (C macro), 291, 527
 kOfxKey_KP_Down (C macro), 291, 527
 kOfxKey_KP_End (C macro), 291, 527
 kOfxKey_KP_Enter (C macro), 290, 526
 kOfxKey_KP_Equal (C macro), 291, 527
 kOfxKey_KP_F1 (C macro), 290, 526
 kOfxKey_KP_F2 (C macro), 290, 526
 kOfxKey_KP_F3 (C macro), 290, 526
 kOfxKey_KP_F4 (C macro), 290, 526
 kOfxKey_KP_Home (C macro), 290, 526
 kOfxKey_KP_Insert (C macro), 291, 527
 kOfxKey_KP_Left (C macro), 290, 526
 kOfxKey_KP_Multiply (C macro), 291, 527
 kOfxKey_KP_Next (C macro), 291, 527
 kOfxKey_KP_Page_Down (C macro), 291, 527
 kOfxKey_KP_Page_Up (C macro), 291, 527
 kOfxKey_KP_Prior (C macro), 291, 527
 kOfxKey_KP_Right (C macro), 291, 527
 kOfxKey_KP_Separator (C macro), 291, 527
 kOfxKey_KP_Space (C macro), 290, 526
 kOfxKey_KP_Subtract (C macro), 291, 527
 kOfxKey_KP_Tab (C macro), 290, 526
 kOfxKey_KP_Up (C macro), 291, 526
 kOfxKey_L (C macro), 298, 533
 kOfxKey_l (C macro), 299, 535
 kOfxKey_L1 (C macro), 292, 528
 kOfxKey_L10 (C macro), 293, 529
 kOfxKey_L2 (C macro), 292, 528
 kOfxKey_L3 (C macro), 293, 529
 kOfxKey_L4 (C macro), 293, 529
 kOfxKey_L5 (C macro), 293, 529
 kOfxKey_L6 (C macro), 293, 529
 kOfxKey_L7 (C macro), 293, 529
 kOfxKey_L8 (C macro), 293, 529
 kOfxKey_L9 (C macro), 293, 529
 kOfxKey_Left (C macro), 289, 525
 kOfxKey_less (C macro), 297, 533

kOfxKey_Linefeed (*C macro*), 287, 523
kOfxKey_M (*C macro*), 298, 534
kOfxKey_m (*C macro*), 299, 535
kOfxKey_macron (*C macro*), 301, 537
kOfxKey_Mae_Koho (*C macro*), 289, 525
kOfxKey_masculine (*C macro*), 301, 537
kOfxKey_Massyo (*C macro*), 289, 525
kOfxKey_Menu (*C macro*), 290, 526
kOfxKey_Meta_L (*C macro*), 295, 531
kOfxKey_Meta_R (*C macro*), 295, 531
kOfxKey_minus (*C macro*), 296, 532
kOfxKey_Mode_switch (*C macro*), 290, 526
kOfxKey_mu (*C macro*), 301, 537
kOfxKey_Muhenkan (*C macro*), 288, 524
kOfxKey_Multi_key (*C macro*), 288, 524
kOfxKey_MultipleCandidate (*C macro*), 288, 524
kOfxKey_multiply (*C macro*), 303, 539
kOfxKey_N (*C macro*), 298, 534
kOfxKey_n (*C macro*), 299, 535
kOfxKey_Next (*C macro*), 289, 525
kOfxKey_nobreakspace (*C macro*), 300, 536
kOfxKey_notsign (*C macro*), 301, 537
kOfxKey_Ntilde (*C macro*), 302, 538
kOfxKey_ntilde (*C macro*), 304, 540
kOfxKey_Num_Lock (*C macro*), 290, 526
kOfxKey_numbersign (*C macro*), 296, 531
kOfxKey_0 (*C macro*), 298, 534
kOfxKey_o (*C macro*), 299, 535
kOfxKey_Oacute (*C macro*), 302, 538
kOfxKey_oacute (*C macro*), 304, 540
kOfxKey_Ocircumflex (*C macro*), 303, 538
kOfxKey_ocircumflex (*C macro*), 304, 540
kOfxKey_Odiaeresis (*C macro*), 303, 539
kOfxKey_odiaeresis (*C macro*), 304, 540
kOfxKey_Ograve (*C macro*), 302, 538
kOfxKey_ograde (*C macro*), 304, 540
kOfxKey_onehalf (*C macro*), 301, 537
kOfxKey_onequarter (*C macro*), 301, 537
kOfxKey_onesuperior (*C macro*), 301, 537
kOfxKey_Obllique (*C macro*), 303, 539
kOfxKey_ordfeminine (*C macro*), 300, 536
kOfxKey_oslash (*C macro*), 304, 540
kOfxKey_Otilde (*C macro*), 303, 539
kOfxKey_otilde (*C macro*), 304, 540
kOfxKey_P (*C macro*), 298, 534
kOfxKey_p (*C macro*), 299, 535
kOfxKey_Page_Down (*C macro*), 289, 525
kOfxKey_Page_Up (*C macro*), 289, 525
kOfxKey_paragraph (*C macro*), 301, 537
kOfxKey_parenleft (*C macro*), 296, 532
kOfxKey_parenright (*C macro*), 296, 532
kOfxKey_Pause (*C macro*), 288, 524
kOfxKey_percent (*C macro*), 296, 532
kOfxKey_period (*C macro*), 296, 532
kOfxKey_periodcentered (*C macro*), 301, 537
kOfxKey_plus (*C macro*), 296, 532
kOfxKey_plusminus (*C macro*), 301, 537
kOfxKey_PreviousCandidate (*C macro*), 288, 524
kOfxKey_Print (*C macro*), 289, 525
kOfxKey_Prior (*C macro*), 289, 525
kOfxKey_Q (*C macro*), 298, 534
kOfxKey_q (*C macro*), 299, 535
kOfxKey_question (*C macro*), 297, 533
kOfxKey_questiondown (*C macro*), 301, 537
kOfxKey_quotedbl (*C macro*), 295, 531
kOfxKey_quoteleft (*C macro*), 299, 534
kOfxKey_quoteright (*C macro*), 296, 532
kOfxKey_R (*C macro*), 298, 534
kOfxKey_r (*C macro*), 299, 535
kOfxKey_R1 (*C macro*), 293, 529
kOfxKey_R10 (*C macro*), 294, 530
kOfxKey_R11 (*C macro*), 294, 530
kOfxKey_R12 (*C macro*), 294, 530
kOfxKey_R13 (*C macro*), 294, 530
kOfxKey_R14 (*C macro*), 295, 531
kOfxKey_R15 (*C macro*), 295, 531
kOfxKey_R2 (*C macro*), 293, 529
kOfxKey_R3 (*C macro*), 294, 529
kOfxKey_R4 (*C macro*), 294, 530
kOfxKey_R5 (*C macro*), 294, 530
kOfxKey_R6 (*C macro*), 294, 530
kOfxKey_R7 (*C macro*), 294, 530
kOfxKey_R8 (*C macro*), 294, 530
kOfxKey_R9 (*C macro*), 294, 530
kOfxKey_Redo (*C macro*), 290, 526
kOfxKey_registered (*C macro*), 301, 537
kOfxKey_Return (*C macro*), 288, 523
kOfxKey_Right (*C macro*), 289, 525
kOfxKey_Romaji (*C macro*), 288, 524
kOfxKey_S (*C macro*), 298, 534
kOfxKey_s (*C macro*), 299, 535
kOfxKey_script_switch (*C macro*), 290, 526
kOfxKey_Scroll_Lock (*C macro*), 288, 524
kOfxKey_section (*C macro*), 300, 536
kOfxKey_Select (*C macro*), 289, 525
kOfxKey_semicolon (*C macro*), 297, 533
kOfxKey_Shift_L (*C macro*), 295, 531
kOfxKey_Shift_Lock (*C macro*), 295, 531
kOfxKey_Shift_R (*C macro*), 295, 531
kOfxKey_SingleCandidate (*C macro*), 288, 524
kOfxKey_slash (*C macro*), 296, 532
kOfxKey_space (*C macro*), 295, 531
kOfxKey_ssharp (*C macro*), 303, 539
kOfxKey_sterling (*C macro*), 300, 536
kOfxKey_Super_L (*C macro*), 295, 531
kOfxKey_Super_R (*C macro*), 295, 531
kOfxKey_Sys_Req (*C macro*), 288, 524
kOfxKey_T (*C macro*), 298, 534

- kOfxKey_t (C macro), 299, 535
- kOfxKey_Tab (C macro), 287, 523
- kOfxKey_THORN (C macro), 303, 539
- kOfxKey_thorn (C macro), 305, 540
- kOfxKey_threequarters (C macro), 301, 537
- kOfxKey_threesuperior (C macro), 301, 537
- kOfxKey_Touroku (C macro), 289, 524
- kOfxKey_twosuperior (C macro), 301, 537
- kOfxKey_U (C macro), 298, 534
- kOfxKey_u (C macro), 300, 535
- kOfxKey_Uacute (C macro), 303, 539
- kOfxKey_uacute (C macro), 304, 540
- kOfxKey_Ucircumflex (C macro), 303, 539
- kOfxKey_ucircumflex (C macro), 304, 540
- kOfxKey_Udiaeresis (C macro), 303, 539
- kOfxKey_udiaeresis (C macro), 304, 540
- kOfxKey_Ugrave (C macro), 303, 539
- kOfxKey_ugrave (C macro), 304, 540
- kOfxKey_underscore (C macro), 298, 534
- kOfxKey_Undo (C macro), 290, 526
- kOfxKey_Unknown (C macro), 287, 523
- kOfxKey_Up (C macro), 289, 525
- kOfxKey_V (C macro), 298, 534
- kOfxKey_v (C macro), 300, 536
- kOfxKey_W (C macro), 298, 534
- kOfxKey_w (C macro), 300, 536
- kOfxKey_X (C macro), 298, 534
- kOfxKey_x (C macro), 300, 536
- kOfxKey_Y (C macro), 298, 534
- kOfxKey_y (C macro), 300, 536
- kOfxKey_Yacute (C macro), 303, 539
- kOfxKey_yacute (C macro), 304, 540
- kOfxKey_ydiaeresis (C macro), 305, 541
- kOfxKey_yen (C macro), 300, 536
- kOfxKey_Z (C macro), 298, 534
- kOfxKey_z (C macro), 300, 536
- kOfxKey_Zen_Koho (C macro), 289, 525
- kOfxKey_Zenkaku (C macro), 288, 524
- kOfxKey_Zenkaku_Hankaku (C macro), 288, 524
- kOfxMemorySuite (C macro), 305, 541
- kOfxMessageError (C macro), 306, 541
- kOfxMessageFatal (C macro), 306, 541
- kOfxMessageLog (C macro), 307, 542
- kOfxMessageMessage (C macro), 307, 541
- kOfxMessageQuestion (C macro), 307, 542
- kOfxMessageSuite (C macro), 306, 541
- kOfxMessageWarning (C macro), 306, 541
- kOfxMultiThreadSuite (C macro), 310, 542
- kOfxOpenCLProgramSuite (C macro), 73, 146, 239, 483, 578
- kOfxOpenGLPropPixelFormatDepth (C macro), 184, 232, 476, 570
- kOfxOpenGLRenderSuite (C macro), 231, 476, 570
- kOfxParamCoordinatesCanonical (C macro), 55, 327, 556
- kOfxParamCoordinatesNormalised (C macro), 55, 327, 556
- kOfxParamDoubleTypeAbsoluteTime (C macro), 52, 326, 555
- kOfxParamDoubleTypeAngle (C macro), 52, 326, 555
- kOfxParamDoubleTypeNormalisedX (C macro), 33, 53, 313, 544
- kOfxParamDoubleTypeNormalisedXAbsolute (C macro), 33, 53, 314, 544
- kOfxParamDoubleTypeNormalisedXY (C macro), 34, 53, 314, 544
- kOfxParamDoubleTypeNormalisedXYAbsolute (C macro), 34, 53, 314, 544
- kOfxParamDoubleTypeNormalisedY (C macro), 34, 53, 313, 544
- kOfxParamDoubleTypeNormalisedYAbsolute (C macro), 34, 53, 314, 544
- kOfxParamDoubleTypePlain (C macro), 52, 326, 555
- kOfxParamDoubleTypeScale (C macro), 52, 326, 555
- kOfxParamDoubleTypeTime (C macro), 52, 326, 555
- kOfxParamDoubleTypeX (C macro), 32, 52, 326, 555
- kOfxParamDoubleTypeXAbsolute (C macro), 33, 52, 326, 555
- kOfxParamDoubleTypeXY (C macro), 33, 52, 326, 555
- kOfxParamDoubleTypeXYAbsolute (C macro), 33, 52, 326, 555
- kOfxParamDoubleTypeY (C macro), 33, 52, 326, 555
- kOfxParamDoubleTypeYAbsolute (C macro), 33, 52, 326, 555
- kOfxParameterSuite (C macro), 317, 546
- kOfxParametricParameterSuite (C macro), 343, 565
- kOfxParamHostPropMaxPages (C macro), 185, 319, 548
- kOfxParamHostPropMaxParameters (C macro), 185, 319, 548
- kOfxParamHostPropPageRowColumnCount (C macro), 185, 319, 548
- kOfxParamHostPropSupportsBooleanAnimation (C macro), 185, 318, 547
- kOfxParamHostPropSupportsChoiceAnimation (C macro), 185, 318, 547
- kOfxParamHostPropSupportsCustomAnimation (C macro), 186, 318, 547
- kOfxParamHostPropSupportsCustomInteract (C macro), 186, 319, 548
- kOfxParamHostPropSupportsParametricAnimation (C macro), 186, 344, 566
- kOfxParamHostPropSupportsStrChoice (C macro), 186, 331, 560
- kOfxParamHostPropSupportsStrChoiceAnimation (C macro), 186, 331, 560
- kOfxParamHostPropSupportsStringAnimation (C

- macro*), 187, 318, 547
- kOfxParamInvalidateAll (*C macro*), 324, 553
- kOfxParamInvalidateValueChange (*C macro*), 324, 553
- kOfxParamInvalidateValueChangeToEnd (*C macro*), 324, 553
- kOfxParamPageSkipColumn (*C macro*), 49, 319, 548
- kOfxParamPageSkipRow (*C macro*), 49, 319, 548
- kOfxParamPropAnimates (*C macro*), 187, 321, 550
- kOfxParamPropCacheInvalidation (*C macro*), 187, 324, 553
- kOfxParamPropCanUndo (*C macro*), 188, 321, 550
- kOfxParamPropChoiceEnum (*C macro*), 188, 330, 559
- kOfxParamPropChoiceOption (*C macro*), 188, 329, 559
- kOfxParamPropChoiceOrder (*C macro*), 188, 330, 559
- kOfxParamPropCustomInterpCallbackV1 (*C macro*), 189, 333, 562
- kOfxParamPropCustomValue (*C macro*), 189, 334, 564
- kOfxParamPropDataPtr (*C macro*), 189, 329, 558
- kOfxParamPropDefault (*C macro*), 190, 324, 553
- kOfxParamPropDefaultCoordinateSystem (*C macro*), 190, 326, 556
- kOfxParamPropDigits (*C macro*), 191, 332, 561
- kOfxParamPropDimensionLabel (*C macro*), 191, 333, 562
- kOfxParamPropDisplayMax (*C macro*), 191, 332, 561
- kOfxParamPropDisplayMin (*C macro*), 191, 332, 561
- kOfxParamPropDoubleType (*C macro*), 192, 325, 554
- kOfxParamPropEnabled (*C macro*), 192, 329, 558
- kOfxParamPropEvaluateOnChange (*C macro*), 193, 323, 552
- kOfxParamPropGroupOpen (*C macro*), 193, 329, 558
- kOfxParamPropHasHostOverlayHandle (*C macro*), 193, 327, 556
- kOfxParamPropHint (*C macro*), 193, 324, 553
- kOfxParamPropIncrement (*C macro*), 193, 332, 561
- kOfxParamPropInteractMinimumSize (*C macro*), 194, 320, 549
- kOfxParamPropInteractPreferredSize (*C macro*), 194, 321, 550
- kOfxParamPropInteractSize (*C macro*), 194, 320, 549
- kOfxParamPropInteractSizeAspect (*C macro*), 194, 320, 549
- kOfxParamPropInteractV1 (*C macro*), 195, 320, 549
- kOfxParamPropInterpolationAmount (*C macro*), 195, 335, 564
- kOfxParamPropInterpolationTime (*C macro*), 195, 334, 564
- kOfxParamPropIsAnimating (*C macro*), 195, 322, 551
- kOfxParamPropIsAutoKeying (*C macro*), 196, 333, 562
- kOfxParamPropMax (*C macro*), 196, 331, 560
- kOfxParamPropMin (*C macro*), 196, 331, 560
- kOfxParamPropPageChild (*C macro*), 196, 328, 557
- kOfxParamPropParametricDimension (*C macro*), 197, 343, 565
- kOfxParamPropParametricInteractBackground (*C macro*), 197, 344, 566
- kOfxParamPropParametricRange (*C macro*), 197, 345, 566
- kOfxParamPropParametricUIColor (*C macro*), 197, 344, 565
- kOfxParamPropParent (*C macro*), 198, 328, 558
- kOfxParamPropPersistant (*C macro*), 198, 322, 551
- kOfxParamPropPluginMayWrite (*C macro*), 198, 322, 551
- kOfxParamPropScriptName (*C macro*), 198, 323, 552
- kOfxParamPropSecret (*C macro*), 199, 323, 552
- kOfxParamPropShowTimeMarker (*C macro*), 199, 328, 557
- kOfxParamPropStringFilePathExists (*C macro*), 334, 563
- kOfxParamPropStringMode (*C macro*), 199, 333, 562
- kOfxParamPropType (*C macro*), 200, 321, 550
- kOfxParamPropUseHostOverlayHandle (*C macro*), 200, 327, 556
- kOfxParamStringIsDirectoryPath (*C macro*), 45, 334, 563
- kOfxParamStringIsFilePath (*C macro*), 45, 334, 563
- kOfxParamStringIsLabel (*C macro*), 45, 334, 563
- kOfxParamStringIsMultiLine (*C macro*), 45, 334, 563
- kOfxParamStringIsRichTextFormat (*C macro*), 45, 334, 563
- kOfxParamStringIsSingleLine (*C macro*), 45, 334, 563
- kOfxParamTypeBoolean (*C macro*), 41, 317, 546
- kOfxParamTypeChoice (*C macro*), 41, 317, 546
- kOfxParamTypeCustom (*C macro*), 41, 318, 547
- kOfxParamTypeDouble (*C macro*), 41, 317, 546
- kOfxParamTypeDouble2D (*C macro*), 41, 317, 546
- kOfxParamTypeDouble3D (*C macro*), 41, 317, 546
- kOfxParamTypeGroup (*C macro*), 42, 318, 547
- kOfxParamTypeInteger (*C macro*), 41, 317, 546
- kOfxParamTypeInteger2D (*C macro*), 41, 317, 546
- kOfxParamTypeInteger3D (*C macro*), 41, 317, 546
- kOfxParamTypePage (*C macro*), 42, 318, 547
- kOfxParamTypeParametric (*C macro*), 42, 343, 565
- kOfxParamTypePushButton (*C macro*), 41, 318, 547
- kOfxParamTypeRGB (*C macro*), 41, 317, 546
- kOfxParamTypeRGBA (*C macro*), 41, 317, 546
- kOfxParamTypeStrChoice (*C macro*), 41, 317, 546
- kOfxParamTypeString (*C macro*), 41, 318, 547
- kOfxPluginPropFilePath (*C macro*), 200, 213, 464
- kOfxPluginPropParamPageOrder (*C macro*), 200, 328, 557

kOfxProgressSuite (*C macro*), 352, 568
 kOfxPropAPIVersion (*C macro*), 200, 213, 464
 kOfxPropChangeReason (*C macro*), 201, 216, 467
 kOfxPropEffectInstance (*C macro*), 201, 217, 468
 kOfxPropertySuite (*C macro*), 355, 569
 kOfxPropHostOSHandle (*C macro*), 201, 217, 468
 kOfxPropIcon (*C macro*), 201, 215, 467
 kOfxPropInstanceData (*C macro*), 202, 214, 465
 kOfxPropIsInteractive (*C macro*), 202, 213, 464
 kOfxPropKeyString (*C macro*), 202, 287, 523
 kOfxPropKeySym (*C macro*), 203, 287, 523
 kOfxPropLabel (*C macro*), 203, 215, 466
 kOfxPropLongLabel (*C macro*), 203, 216, 467
 kOfxPropName (*C macro*), 203, 214, 465
 kOfxPropParamSetNeedsSyncing (*C macro*), 204, 321, 550
 kOfxPropPluginDescription (*C macro*), 204, 215, 466
 kOfxPropShortLabel (*C macro*), 204, 216, 467
 kOfxPropTime (*C macro*), 205, 213, 464
 kOfxPropType (*C macro*), 205, 214, 465
 kOfxPropVersion (*C macro*), 205, 214, 465
 kOfxPropVersionLabel (*C macro*), 205, 215, 466
 kOfxStatErrBadHandle (*C macro*), 218, 470, 585
 kOfxStatErrBadIndex (*C macro*), 219, 470, 585
 kOfxStatErrExists (*C macro*), 218, 469, 584
 kOfxStatErrFatal (*C macro*), 218, 469, 584
 kOfxStatErrFormat (*C macro*), 218, 469, 585
 kOfxStatErrImageFormat (*C macro*), 270, 512
 kOfxStatErrMemory (*C macro*), 218, 469, 585
 kOfxStatErrMissingHostFeature (*C macro*), 218, 469, 584
 kOfxStatErrUnknown (*C macro*), 218, 469, 584
 kOfxStatErrUnsupported (*C macro*), 218, 469, 584
 kOfxStatErrValue (*C macro*), 219, 470, 585
 kOfxStatFailed (*C macro*), 218, 469, 584
 kOfxStatGLOutOfMemory (*C macro*), 231, 476, 570, 585
 kOfxStatGLRenderFailed (*C macro*), 231, 476, 570, 585
 kOfxStatGPUOutOfMemory (*C macro*), 231, 475, 570, 585
 kOfxStatGPURenderFailed (*C macro*), 231, 476, 570, 585
 kOfxStatOK (*C macro*), 218, 469, 584
 kOfxStatReplyDefault (*C macro*), 219, 470, 585
 kOfxStatReplyNo (*C macro*), 219, 470, 585
 kOfxStatReplyYes (*C macro*), 219, 470, 585
 kOfxTimeLineSuite (*C macro*), 362, 569
 kOfxTypeClip (*C macro*), 243, 485
 kOfxTypeImage (*C macro*), 243, 485
 kOfxTypeImageEffect (*C macro*), 243, 484
 kOfxTypeImageEffectHost (*C macro*), 243, 484
 kOfxTypeImageEffectInstance (*C macro*), 243, 485
 kOfxTypeParameter (*C macro*), 317, 546

kOfxTypeParameterInstance (*C macro*), 317, 546

O

OfxDialogSuiteV1 (*C++ struct*), 225, 364, 411
 OfxDialogSuiteV1 (*C++ type*), 225, 473
 OfxDialogSuiteV1::NotifyRedrawPending (*C++ member*), 226, 364, 412
 OfxDialogSuiteV1::RequestDialog (*C++ member*), 226, 364, 412
 OfxDrawContextHandle (*C++ type*), 227, 473
 OfxDrawLineStipplePattern (*C++ enum*), 150, 227, 474
 OfxDrawLineStipplePattern (*C++ type*), 227, 473
 OfxDrawLineStipplePattern::kOfxDrawLineStipplePatternAltDash (*C++ enumerator*), 150, 228, 474
 OfxDrawLineStipplePattern::kOfxDrawLineStipplePatternDash (*C++ enumerator*), 150, 228, 474
 OfxDrawLineStipplePattern::kOfxDrawLineStipplePatternDot (*C++ enumerator*), 150, 228, 474
 OfxDrawLineStipplePattern::kOfxDrawLineStipplePatternDotDash (*C++ enumerator*), 150, 228, 474
 OfxDrawLineStipplePattern::kOfxDrawLineStipplePatternSolid (*C++ enumerator*), 150, 227, 474
 OfxDrawPrimitive (*C++ enum*), 150, 228, 474
 OfxDrawPrimitive (*C++ type*), 227, 473
 OfxDrawPrimitive::kOfxDrawPrimitiveEllipse (*C++ enumerator*), 150, 228, 475
 OfxDrawPrimitive::kOfxDrawPrimitiveLineLoop (*C++ enumerator*), 150, 228, 474
 OfxDrawPrimitive::kOfxDrawPrimitiveLines (*C++ enumerator*), 150, 228, 474
 OfxDrawPrimitive::kOfxDrawPrimitiveLineStrip (*C++ enumerator*), 150, 228, 474
 OfxDrawPrimitive::kOfxDrawPrimitivePolygon (*C++ enumerator*), 150, 228, 475
 OfxDrawPrimitive::kOfxDrawPrimitiveRectangle (*C++ enumerator*), 150, 228, 475
 OfxDrawSuiteV1 (*C++ struct*), 147, 229, 364, 412
 OfxDrawSuiteV1 (*C++ type*), 227, 473
 OfxDrawSuiteV1::draw (*C++ member*), 148, 230, 365, 413
 OfxDrawSuiteV1::drawText (*C++ member*), 149, 230, 366, 414
 OfxDrawSuiteV1::getColour (*C++ member*), 147, 229, 364, 412
 OfxDrawSuiteV1::setColour (*C++ member*), 147, 229, 365, 412
 OfxDrawSuiteV1::setLineStipple (*C++ member*), 148, 230, 365, 413
 OfxDrawSuiteV1::setLineWidth (*C++ member*), 148, 229, 365, 413
 OfxDrawTextAlignment (*C++ enum*), 228, 475
 OfxDrawTextAlignment (*C++ type*), 227, 473

OfxDrawTextAlignment::kOfxDrawTextAlignmentBaseline *member*), 110, 117, 271, 370, 417
 (C++ *enumerator*), 228, 475 OfxImageEffectSuiteV1::imageMemoryAlloc
 OfxDrawTextAlignment::kOfxDrawTextAlignmentBottom (C++ *member*), 114, 121, 275, 373, 421
 (C++ *enumerator*), 228, 475 OfxImageEffectSuiteV1::imageMemoryFree (C++
 OfxDrawTextAlignment::kOfxDrawTextAlignmentCenterH *member*), 114, 121, 275, 374, 421
 (C++ *enumerator*), 228, 475 OfxImageEffectSuiteV1::imageMemoryLock (C++
 OfxDrawTextAlignment::kOfxDrawTextAlignmentCenterV *member*), 114, 122, 275, 374, 421
 (C++ *enumerator*), 228, 475 OfxImageEffectSuiteV1::imageMemoryUnlock
 OfxDrawTextAlignment::kOfxDrawTextAlignmentLeft (C++ *member*), 115, 122, 276, 375, 422
 (C++ *enumerator*), 228, 475 OfxImageMemoryHandle (C++ *type*), 271, 513
 OfxDrawTextAlignment::kOfxDrawTextAlignmentRight (C++ *member*), 115, 122, 276, 375, 422
 (C++ *enumerator*), 228, 475 OfxInteractHandle (C++ *type*), 21, 286, 522
 OfxDrawTextAlignment::kOfxDrawTextAlignmentTop (C++ *member*), 115, 122, 276, 375, 422
 (C++ *enumerator*), 228, 475 OfxInteractSuiteV1 (C++ *struct*), 137, 286, 375, 422
 OfxInteractSuiteV1 (C++ *type*), 286, 522
 OfxExport (C *macro*), 206, 457 OfxInteractSuiteV1::interactGetPropertySet
 (C++ *member*), 137, 286, 375, 423
 OfxGetNumberOfPlugins (C++ *function*), 6, 221, 472 OfxInteractSuiteV1::interactRedraw (C++ *mem-*
 OfxGetPlugin (C++ *function*), 6, 221, 472 *ber*), 137, 286, 375, 423
 OfxHost (C++ *struct*), 9, 221, 367, 414 OfxInteractSuiteV1::interactSwapBuffers
 OfxHost (C++ *type*), 219, 470 (C++ *member*), 137, 286, 375, 423
 OfxHost::fetchSuite (C++ *member*), 9, 221, 367,
 414 OfxMemorySuiteV1 (C++ *struct*), 133, 305, 376, 423
 OfxHost::host (C++ *member*), 9, 221, 367, 414 OfxMemorySuiteV1 (C++ *type*), 305, 541
 OfxImageClipHandle (C++ *type*), 19, 271, 513 OfxMemorySuiteV1::memoryAlloc (C++ *member*),
 OfxImageEffectHandle (C++ *type*), 17, 271, 513 134, 306, 376, 423
 OfxImageEffectOpenGLRenderSuiteV1 (C++ *struct*),
 139, 239, 367, 415 OfxMemorySuiteV1::memoryFree (C++ *member*),
 OfxImageEffectOpenGLRenderSuiteV1 (C++ *type*),
 239, 483, 573 134, 306, 376, 423
 OfxImageEffectOpenGLRenderSuiteV1::clipFreeTexture
 (C++ *member*), 141, 240, 369, 416 OfxMessageSuiteV1 (C++ *struct*), 137, 307, 377, 424
 OfxImageEffectOpenGLRenderSuiteV1::clipLoadTexture
 (C++ *member*), 140, 239, 367, 415 OfxMessageSuiteV1 (C++ *type*), 307, 542
 OfxImageEffectOpenGLRenderSuiteV1::flushResources
 (C++ *member*), 141, 241, 369, 417 OfxMessageSuiteV1::message (C++ *member*), 137,
 OfxImageEffectSuiteV1 (C++ *struct*), 110, 117, 271,
 370, 417 308, 377, 424
 OfxImageEffectSuiteV1 (C++ *type*), 271, 513 OfxMessageSuiteV2 (C++ *struct*), 138, 308, 377, 424
 OfxImageEffectSuiteV1::abort (C++ *member*),
 113, 121, 274, 373, 420 OfxMessageSuiteV2 (C++ *type*), 307, 542
 OfxImageEffectSuiteV1::clipDefine (C++ *mem-*
ber), 110, 118, 272, 370, 418 OfxMessageSuiteV2::clearPersistentMessage
 OfxImageEffectSuiteV1::clipGetHandle (C++
member), 111, 118, 272, 371, 418 (C++ *member*), 139, 309, 379, 426
 OfxImageEffectSuiteV1::clipGetImage (C++
member), 112, 119, 273, 371, 419 OfxMessageSuiteV2::message (C++ *member*), 138,
 OfxImageEffectSuiteV1::clipGetPropertySet
 (C++ *member*), 111, 119, 272, 371, 419 308, 378, 425
 OfxImageEffectSuiteV1::clipGetRegionOfDefinition
 (C++ *member*), 113, 120, 274, 372, 420 OfxMessageSuiteV2::setPersistentMessage
 OfxImageEffectSuiteV1::clipReleaseImage
 (C++ *member*), 113, 120, 274, 372, 420 (C++ *member*), 138, 309, 378, 425
 OfxImageEffectSuiteV1::getParamSet (C++ *mem-*
ber), 110, 117, 271, 370, 417 OfxMultiThreadSuiteV1 (C++ *struct*), 134, 310, 379,
 OfxImageEffectSuiteV1::getPropertySet (C++
 426
 OfxMultiThreadSuiteV1 (C++ *type*), 310, 543
 OfxMultiThreadSuiteV1::multiThread (C++ *mem-*
 ber), 134, 310, 379, 426
 OfxMultiThreadSuiteV1::multiThreadIndex
 (C++ *member*), 135, 311, 380, 427
 OfxMultiThreadSuiteV1::multiThreadIsSpawnedThread
 (C++ *member*), 135, 311, 380, 427
 OfxMultiThreadSuiteV1::multiThreadNumCPUs
 (C++ *member*), 135, 311, 380, 427
 OfxMultiThreadSuiteV1::mutexCreate (C++ *mem-*
 ber), 135, 311, 380, 427
 OfxMultiThreadSuiteV1::mutexDestroy (C++
 member), 136, 312, 380, 427
 OfxMultiThreadSuiteV1::mutexLock (C++ *mem-*
 ber), 136, 312, 381, 428

OfxMultiThreadSuiteV1::mutexTryLock (C++ member), 136, 312, 381, 428
 OfxMultiThreadSuiteV1::mutexUnLock (C++ member), 136, 312, 381, 428
 OfxMutexHandle (C++ type), 310, 543
 OfxOpenCLProgramSuiteV1 (C++ struct), 73, 146, 241, 382, 428
 OfxOpenCLProgramSuiteV1 (C++ type), 73, 146, 239, 483, 578
 OfxOpenCLProgramSuiteV1::compileProgram (C++ member), 73, 147, 242, 382, 429
 OfxParameterSuiteV1 (C++ struct), 123, 336, 382, 429
 OfxParameterSuiteV1 (C++ type), 336, 565
 OfxParameterSuiteV1::paramCopy (C++ member), 129, 342, 388, 435
 OfxParameterSuiteV1::paramDefine (C++ member), 125, 338, 384, 431
 OfxParameterSuiteV1::paramDeleteAllKeys (C++ member), 124, 337, 383, 430
 OfxParameterSuiteV1::paramDeleteKey (C++ member), 124, 337, 383, 430
 OfxParameterSuiteV1::paramEditBegin (C++ member), 129, 342, 389, 435
 OfxParameterSuiteV1::paramEditEnd (C++ member), 130, 343, 389, 436
 OfxParameterSuiteV1::paramGetDerivative (C++ member), 127, 340, 386, 433
 OfxParameterSuiteV1::paramGetHandle (C++ member), 125, 338, 384, 431
 OfxParameterSuiteV1::paramGetIntegral (C++ member), 127, 340, 387, 433
 OfxParameterSuiteV1::paramGetKeyIndex (C++ member), 124, 337, 383, 429
 OfxParameterSuiteV1::paramGetKeyTime (C++ member), 123, 336, 382, 429
 OfxParameterSuiteV1::paramGetNumKeys (C++ member), 123, 336, 382, 429
 OfxParameterSuiteV1::paramGetPropertySet (C++ member), 126, 339, 385, 432
 OfxParameterSuiteV1::paramGetValue (C++ member), 126, 339, 385, 432
 OfxParameterSuiteV1::paramGetValueAtTime (C++ member), 127, 340, 386, 433
 OfxParameterSuiteV1::paramSetGetPropertySet (C++ member), 125, 338, 385, 431
 OfxParameterSuiteV1::paramSetValue (C++ member), 128, 341, 387, 434
 OfxParameterSuiteV1::paramSetValueAtTime (C++ member), 128, 341, 388, 434
 OfxParametricParameterSuiteV1 (C++ struct), 130, 345, 390, 436
 OfxParametricParameterSuiteV1 (C++ type), 345, 567
 OfxParametricParameterSuiteV1::parametricParamAddControlP (C++ member), 132, 347, 392, 438
 OfxParametricParameterSuiteV1::parametricParamDeleteAllCor (C++ member), 133, 348, 392, 439
 OfxParametricParameterSuiteV1::parametricParamDeleteContro (C++ member), 133, 348, 392, 439
 OfxParametricParameterSuiteV1::parametricParamGetNControlP (C++ member), 131, 346, 390, 437
 OfxParametricParameterSuiteV1::parametricParamGetNthContro (C++ member), 131, 346, 391, 437
 OfxParametricParameterSuiteV1::parametricParamGetValue (C++ member), 131, 346, 390, 437
 OfxParametricParameterSuiteV1::parametricParamSetNthContro (C++ member), 132, 347, 391, 438
 OfxParamHandle (C++ type), 19, 335, 564
 OfxParamSetHandle (C++ type), 335, 564
 OfxPlugin (C++ struct), 221, 393, 439
 OfxPlugin (C++ type), 219, 471
 OfxPlugin::apiVersion (C++ member), 222, 393, 440
 OfxPlugin::mainEntry (C++ member), 222, 394, 440
 OfxPlugin::pluginApi (C++ member), 222, 393, 440
 OfxPlugin::pluginIdentifier (C++ member), 222, 393, 440
 OfxPlugin::pluginVersionMajor (C++ member), 222, 393, 440
 OfxPlugin::pluginVersionMinor (C++ member), 222, 393, 440
 OfxPlugin::setHost (C++ member), 222, 393, 440
 OfxPointD (C++ struct), 223, 394, 441
 OfxPointD (C++ type), 220, 471
 OfxPointD::x (C++ member), 224, 394, 441
 OfxPointD::y (C++ member), 224, 394, 441
 OfxPointI (C++ struct), 223, 394, 441
 OfxPointI (C++ type), 220, 471
 OfxPointI::x (C++ member), 223, 394, 441
 OfxPointI::y (C++ member), 223, 394, 441
 OfxProgressSuiteV1 (C++ struct), 116, 352, 394, 441
 OfxProgressSuiteV1 (C++ type), 352, 568
 OfxProgressSuiteV1::progressEnd (C++ member), 117, 353, 395, 442
 OfxProgressSuiteV1::progressStart (C++ member), 116, 353, 395, 442
 OfxProgressSuiteV1::progressUpdate (C++ member), 116, 353, 395, 442
 OfxProgressSuiteV2 (C++ struct), 354, 396, 442
 OfxProgressSuiteV2 (C++ type), 352, 568
 OfxProgressSuiteV2::progressEnd (C++ member), 354, 397, 443
 OfxProgressSuiteV2::progressStart (C++ member), 354, 396, 443
 OfxProgressSuiteV2::progressUpdate (C++ member), 354, 396, 443
 OfxPropertySetHandle (C++ type), 4, 219, 470

- OfxPropertySuiteV1 (C++ *struct*), 103, 355, 397, 444
OfxPropertySuiteV1 (C++ *type*), 355, 569
OfxPropertySuiteV1::propGetDimension (C++ *member*), 109, 362, 404, 450
OfxPropertySuiteV1::propGetDouble (C++ *member*), 107, 359, 401, 448
OfxPropertySuiteV1::propGetDoubleN (C++ *member*), 108, 361, 403, 449
OfxPropertySuiteV1::propGetInt (C++ *member*), 107, 360, 401, 448
OfxPropertySuiteV1::propGetIntN (C++ *member*), 109, 361, 403, 450
OfxPropertySuiteV1::propGetPointer (C++ *member*), 106, 358, 400, 447
OfxPropertySuiteV1::propGetPointerN (C++ *member*), 108, 360, 402, 448
OfxPropertySuiteV1::propGetString (C++ *member*), 106, 359, 401, 447
OfxPropertySuiteV1::propGetStringN (C++ *member*), 108, 360, 402, 449
OfxPropertySuiteV1::propReset (C++ *member*), 109, 361, 403, 450
OfxPropertySuiteV1::propSetDouble (C++ *member*), 104, 356, 398, 445
OfxPropertySuiteV1::propSetDoubleN (C++ *member*), 105, 358, 400, 446
OfxPropertySuiteV1::propSetInt (C++ *member*), 104, 356, 398, 445
OfxPropertySuiteV1::propSetIntN (C++ *member*), 106, 358, 400, 447
OfxPropertySuiteV1::propSetPointer (C++ *member*), 103, 355, 397, 444
OfxPropertySuiteV1::propSetPointerN (C++ *member*), 104, 357, 399, 445
OfxPropertySuiteV1::propSetString (C++ *member*), 103, 356, 398, 444
OfxPropertySuiteV1::propSetStringN (C++ *member*), 105, 357, 399, 446
OfxRangeD (C++ *struct*), 223, 407, 450
OfxRangeD (C++ *type*), 220, 471
OfxRangeD::max (C++ *member*), 223, 407, 451
OfxRangeD::min (C++ *member*), 223, 407, 451
OfxRangeI (C++ *struct*), 223, 407, 451
OfxRangeI (C++ *type*), 220, 471
OfxRangeI::max (C++ *member*), 223, 407, 451
OfxRangeI::min (C++ *member*), 223, 407, 451
OfxRectD (C++ *struct*), 30, 224, 408, 451
OfxRectD (C++ *type*), 220, 471
OfxRectD::x1 (C++ *member*), 30, 225, 408, 451
OfxRectD::x2 (C++ *member*), 30, 225, 408, 451
OfxRectD::y1 (C++ *member*), 30, 225, 408, 451
OfxRectD::y2 (C++ *member*), 30, 225, 408, 451
OfxRectI (C++ *struct*), 224, 408, 451
OfxRectI (C++ *type*), 220, 471
OfxRectI::x1 (C++ *member*), 224, 409, 452
OfxRectI::x2 (C++ *member*), 224, 409, 452
OfxRectI::y1 (C++ *member*), 224, 409, 452
OfxRectI::y2 (C++ *member*), 224, 409, 452
OfxRGBAColourB (C++ *struct*), 349, 404, 452
OfxRGBAColourB (C++ *type*), 349, 567
OfxRGBAColourB::a (C++ *member*), 349, 404, 452
OfxRGBAColourB::b (C++ *member*), 349, 404, 452
OfxRGBAColourB::g (C++ *member*), 349, 404, 452
OfxRGBAColourB::r (C++ *member*), 349, 404, 452
OfxRGBAColourD (C++ *struct*), 350, 404, 452
OfxRGBAColourD (C++ *type*), 349, 567
OfxRGBAColourD::a (C++ *member*), 350, 405, 453
OfxRGBAColourD::b (C++ *member*), 350, 405, 453
OfxRGBAColourD::g (C++ *member*), 350, 405, 453
OfxRGBAColourD::r (C++ *member*), 350, 405, 453
OfxRGBAColourF (C++ *struct*), 350, 405, 453
OfxRGBAColourF (C++ *type*), 349, 567
OfxRGBAColourF::a (C++ *member*), 350, 405, 453
OfxRGBAColourF::b (C++ *member*), 350, 405, 453
OfxRGBAColourF::g (C++ *member*), 350, 405, 453
OfxRGBAColourF::r (C++ *member*), 350, 405, 453
OfxRGBAColourS (C++ *struct*), 349, 405, 453
OfxRGBAColourS (C++ *type*), 349, 567
OfxRGBAColourS::a (C++ *member*), 350, 405, 453
OfxRGBAColourS::b (C++ *member*), 350, 405, 453
OfxRGBAColourS::g (C++ *member*), 350, 405, 453
OfxRGBAColourS::r (C++ *member*), 350, 405, 453
OfxRGBColourB (C++ *struct*), 350, 406, 453
OfxRGBColourB (C++ *type*), 349, 567
OfxRGBColourB::b (C++ *member*), 351, 406, 454
OfxRGBColourB::g (C++ *member*), 351, 406, 454
OfxRGBColourB::r (C++ *member*), 351, 406, 454
OfxRGBColourD (C++ *struct*), 351, 406, 454
OfxRGBColourD (C++ *type*), 349, 568
OfxRGBColourD::b (C++ *member*), 351, 406, 454
OfxRGBColourD::g (C++ *member*), 351, 406, 454
OfxRGBColourD::r (C++ *member*), 351, 406, 454
OfxRGBColourF (C++ *struct*), 351, 406, 454
OfxRGBColourF (C++ *type*), 349, 568
OfxRGBColourF::b (C++ *member*), 351, 406, 454
OfxRGBColourF::g (C++ *member*), 351, 406, 454
OfxRGBColourF::r (C++ *member*), 351, 406, 454
OfxRGBColourS (C++ *struct*), 351, 407, 454
OfxRGBColourS (C++ *type*), 349, 567
OfxRGBColourS::b (C++ *member*), 351, 407, 454
OfxRGBColourS::g (C++ *member*), 351, 407, 454
OfxRGBColourS::r (C++ *member*), 351, 407, 454
OfxSetHost (C++ *function*), 6, 221, 472
OfxStandardColour (C++ *enum*), 149, 227, 474
OfxStandardColour (C++ *type*), 227, 473
OfxStandardColour::kOfxStandardColourOverlayActive (C++ *enumerator*), 149, 227, 474

OfxStandardColour::kOfxStandardColourOverlayBackground
 (C++ *enumerator*), 149, 227, 474
 OfxStandardColour::kOfxStandardColourOverlayDeselected
 (C++ *enumerator*), 149, 227, 474
 OfxStandardColour::kOfxStandardColourOverlayMarqueeBG
 (C++ *enumerator*), 150, 227, 474
 OfxStandardColour::kOfxStandardColourOverlayMarqueeFG
 (C++ *enumerator*), 150, 227, 474
 OfxStandardColour::kOfxStandardColourOverlaySelected
 (C++ *enumerator*), 149, 227, 474
 OfxStandardColour::kOfxStandardColourOverlayText
 (C++ *enumerator*), 150, 227, 474
 OfxStatus (C++ *type*), 219, 470, 584
 OfxTime (C++ *type*), 220, 471
 OfxTimeLineSuiteV1 (C++ *struct*), 362, 409, 454
 OfxTimeLineSuiteV1 (C++ *type*), 362, 569
 OfxTimeLineSuiteV1::getTime (C++ *member*), 363,
 409, 455
 OfxTimeLineSuiteV1::getTimeBounds (C++ *mem-
 ber*), 363, 410, 455
 OfxTimeLineSuiteV1::gotoTime (C++ *member*),
 363, 409, 455
 OfxYUVAColourB (C++ *struct*), 315, 410, 456
 OfxYUVAColourB (C++ *type*), 315, 545
 OfxYUVAColourB::a (C++ *member*), 315, 410, 456
 OfxYUVAColourB::u (C++ *member*), 315, 410, 456
 OfxYUVAColourB::v (C++ *member*), 315, 410, 456
 OfxYUVAColourB::y (C++ *member*), 315, 410, 456
 OfxYUVAColourF (C++ *struct*), 316, 410, 456
 OfxYUVAColourF (C++ *type*), 315, 545
 OfxYUVAColourF::a (C++ *member*), 316, 411, 456
 OfxYUVAColourF::u (C++ *member*), 316, 411, 456
 OfxYUVAColourF::v (C++ *member*), 316, 411, 456
 OfxYUVAColourF::y (C++ *member*), 316, 411, 456
 OfxYUVAColourS (C++ *struct*), 315, 411, 456
 OfxYUVAColourS (C++ *type*), 315, 545
 OfxYUVAColourS::a (C++ *member*), 316, 411, 457
 OfxYUVAColourS::u (C++ *member*), 316, 411, 457
 OfxYUVAColourS::v (C++ *member*), 316, 411, 457
 OfxYUVAColourS::y (C++ *member*), 316, 411, 457