Android's SurfaceFlinger

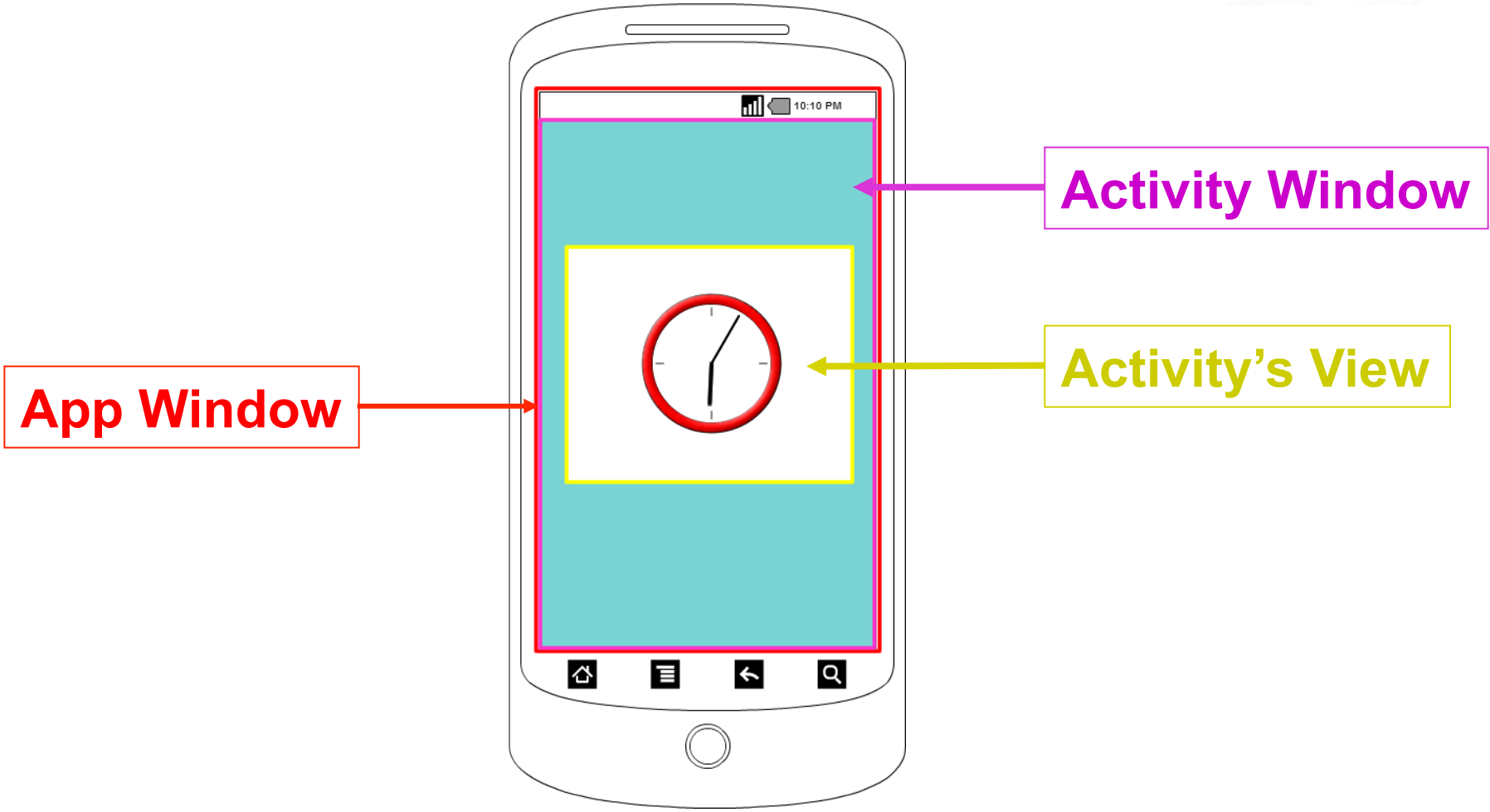
David Lau • China

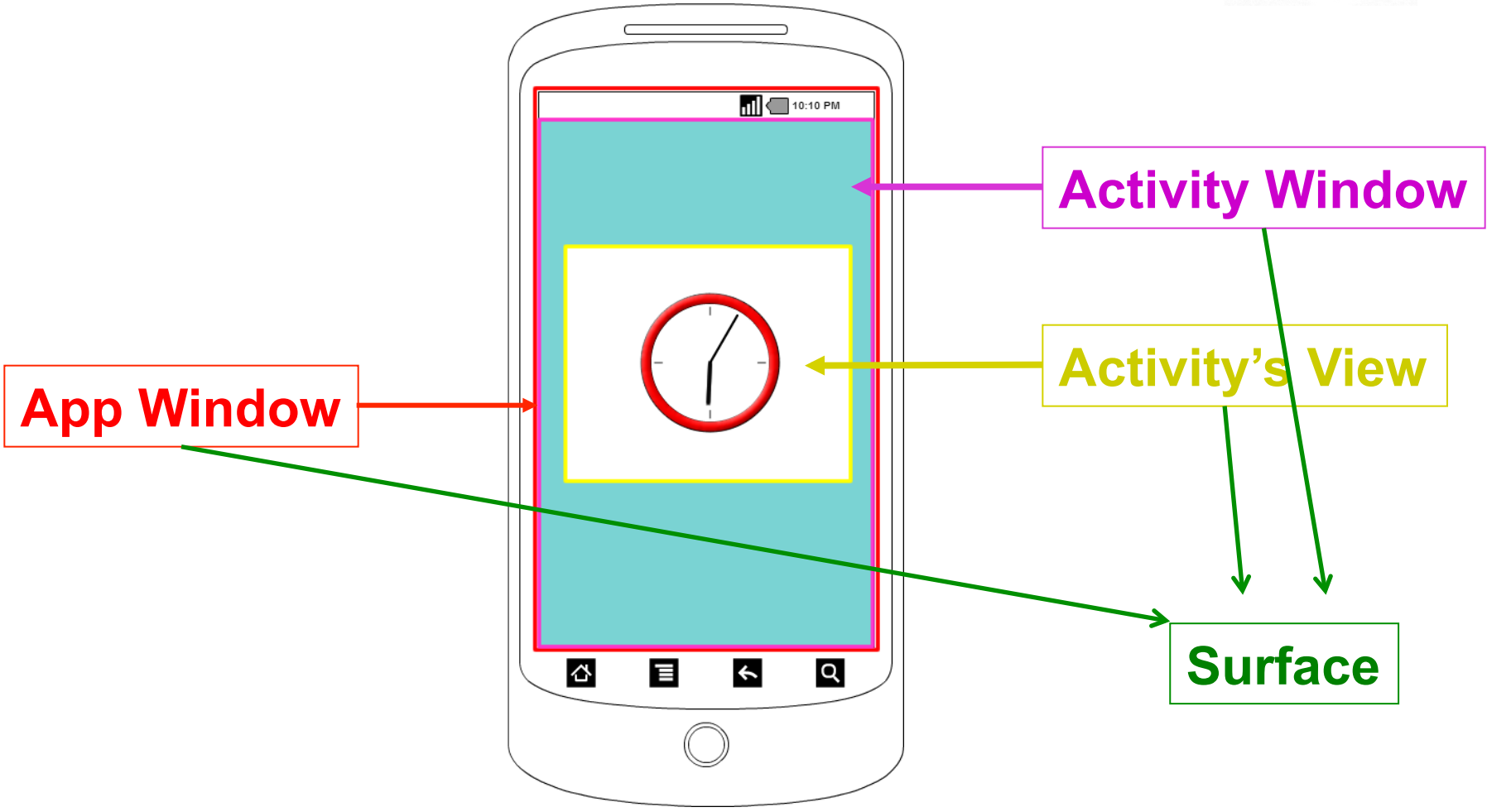
Email: zhiyong.liu@aliyun.com Latest Update：2013-09-08

皮影戏 http://dv1.dhbc.net/ziyuanku1/news_shows.aspx?Cid=91&Nid=374

Overview of the SurfaceFlinger

Android's SurfaceFlinger


10:10 PM
Activity Window
Activity's View
App Window


10:10 PM
Activity Window
Activity's View
App Window
Surface

Surface

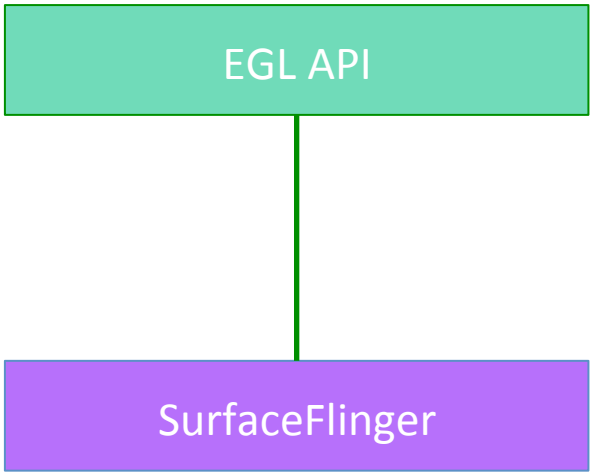
- A Surface in Android corresponds to an offscreen buffer into which application renders its content. From application point of view, each application may correspond to one or more graphical interfaces. Each interface can be regarded as surface.
- Each surface has its position, size, content and other elements.

SurfaceFlinger

- It is a system-wide surface composer function which resides in android framework. **It takes data(which is surface) from different application which could be 2D or 3D and finally combine it to obtain a main surface which will be fed to memory**(which is framebuffer).
- Surfaceflinger synthesize all the surface according **to their position, size and other parameters**, although this synthesis is done by OpenGL(which is invoked by Surfaceflinger), but we need Surfaceflinger to calculate the relevant parameters like overlapping function.

SurfaceFlinger

- The SurfaceFlinger is Android‘s **window compositor**. Each window is a OpenGL texture. The SurfaceFlinger just blends these OpenGL textures one upon the other. So SurfaceFlinger is completely implemented using OpenGL APIs.



HardwareComposer

- Part of HAL, it is used by SurfaceFlinger to **composite surfaces** to the screen. The hardware composer abstracts things like overlays and 2D blitters and helps offload some things that would normally be done by OpenGL. This is done by using 3D GPU or a 2D graphics engine.

Gralloc

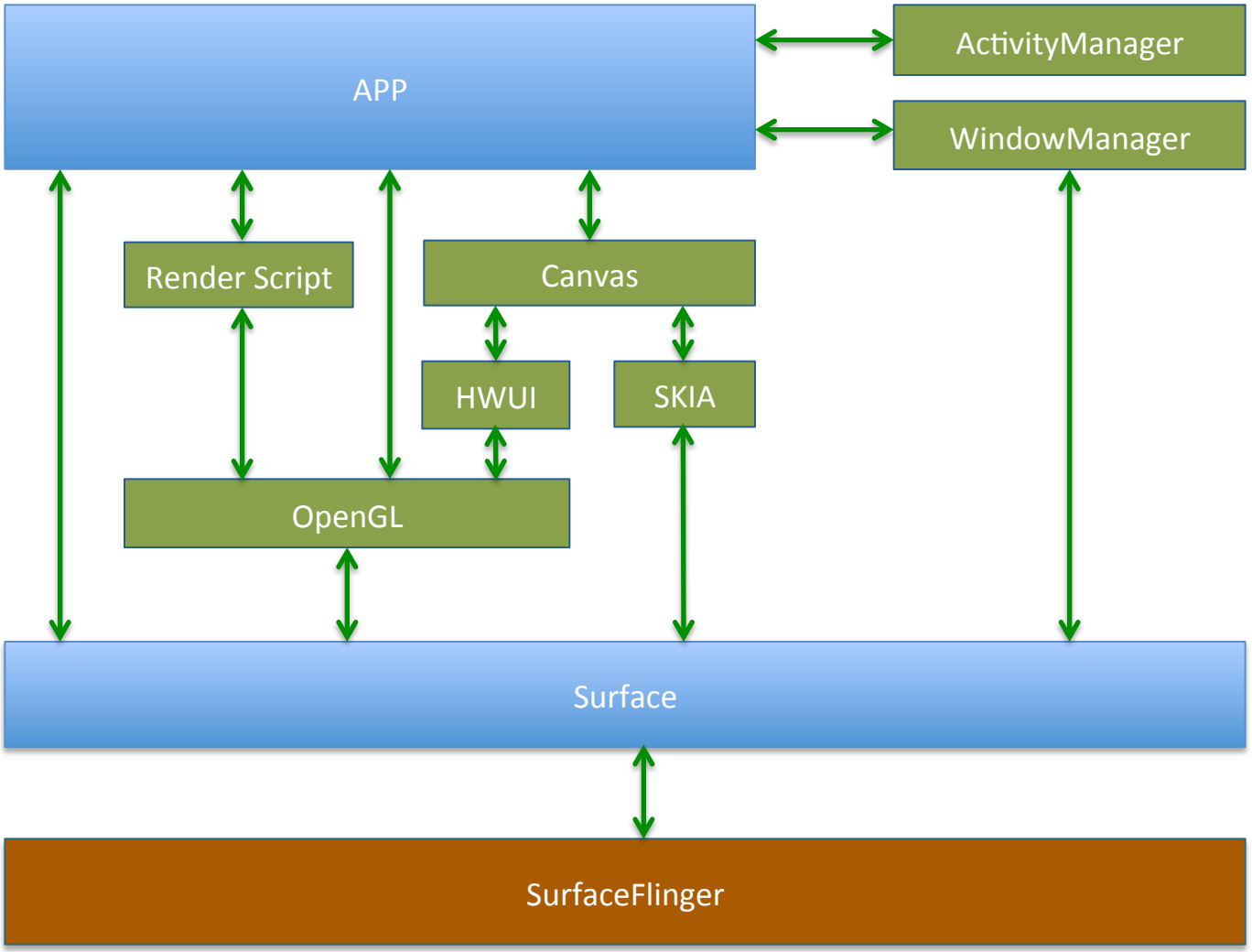
Part of HAL. Gralloc stands for Graphics memory allocator. It has two parts：

1. First provide pmem interface, which is responsible for contiguous memory allocation.
2. Other is responsible for framebuffer refresh where UI actually put framebuffer data.

Display

Physical display device, Current version of SurfaceFlinger supports only one display device which corresponds to display hardware.

App-Surface-SurfaceFlinger



From :linux

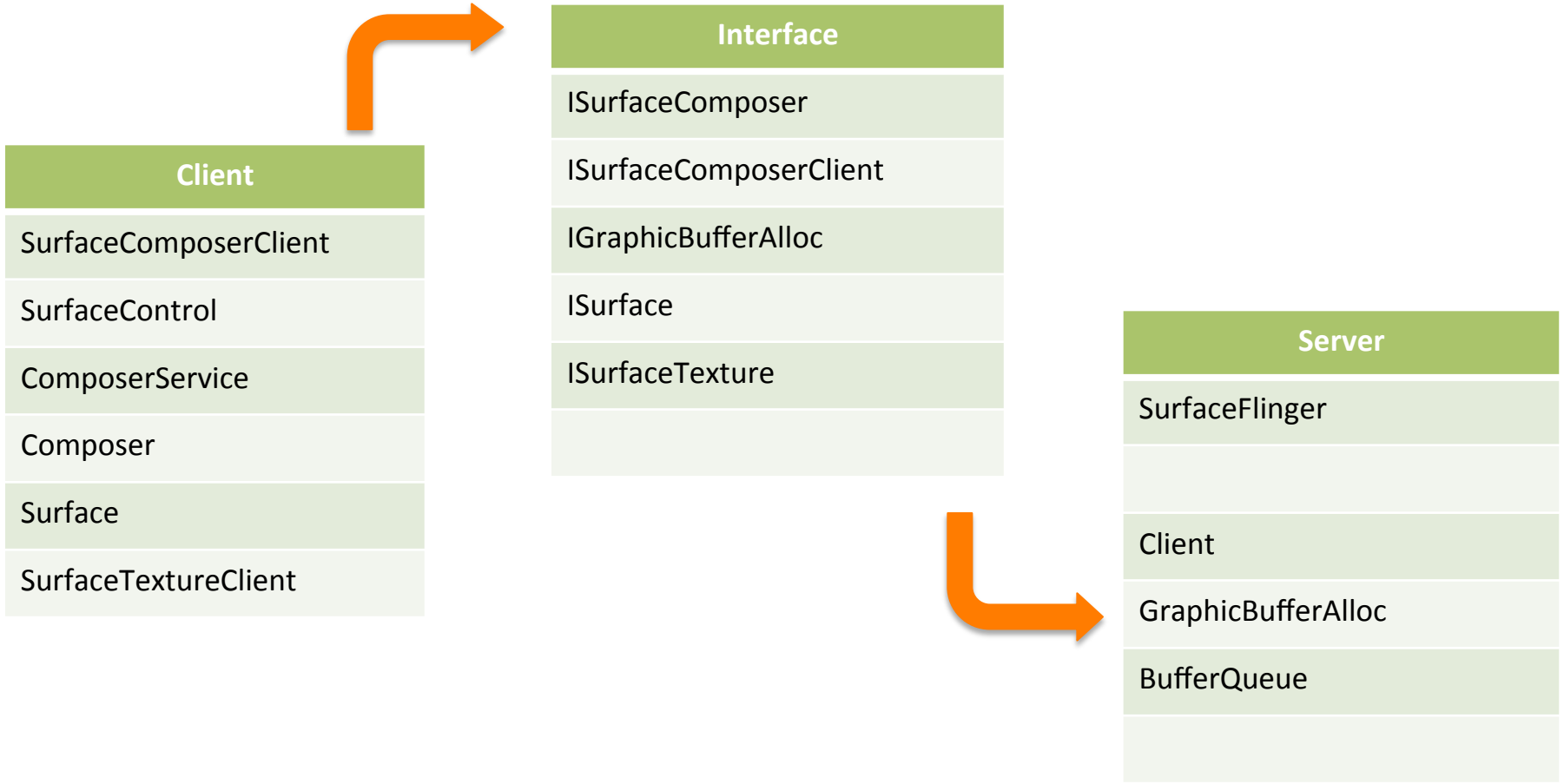
SurfaceFlinger's Three Tasks

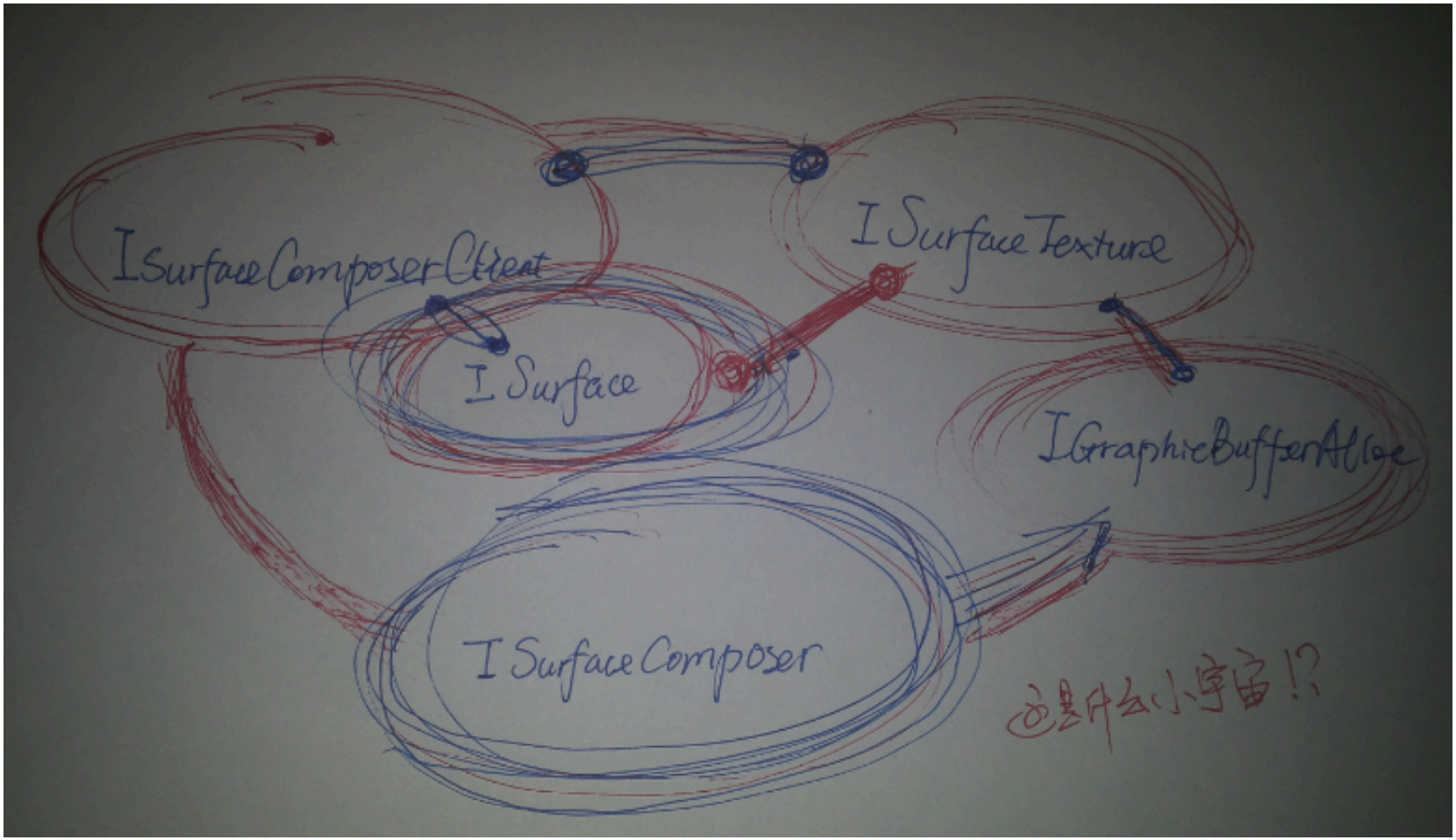
- Creates a new DisplayDevice, which is used to establish a FramebufferNativeWindow to determine the data output device interface
- Initialize OpenGL, as this is the one which synthesize.
- Create main surface, on which all surfaces will be merged.

Functions

- Can combine 2D/3D surfaces and surfaces from multiple applications
- Surfaces passed as buffers via Binder IPC calls
- Can use OpenGL ES and 2D hardware accelerator for its compositions

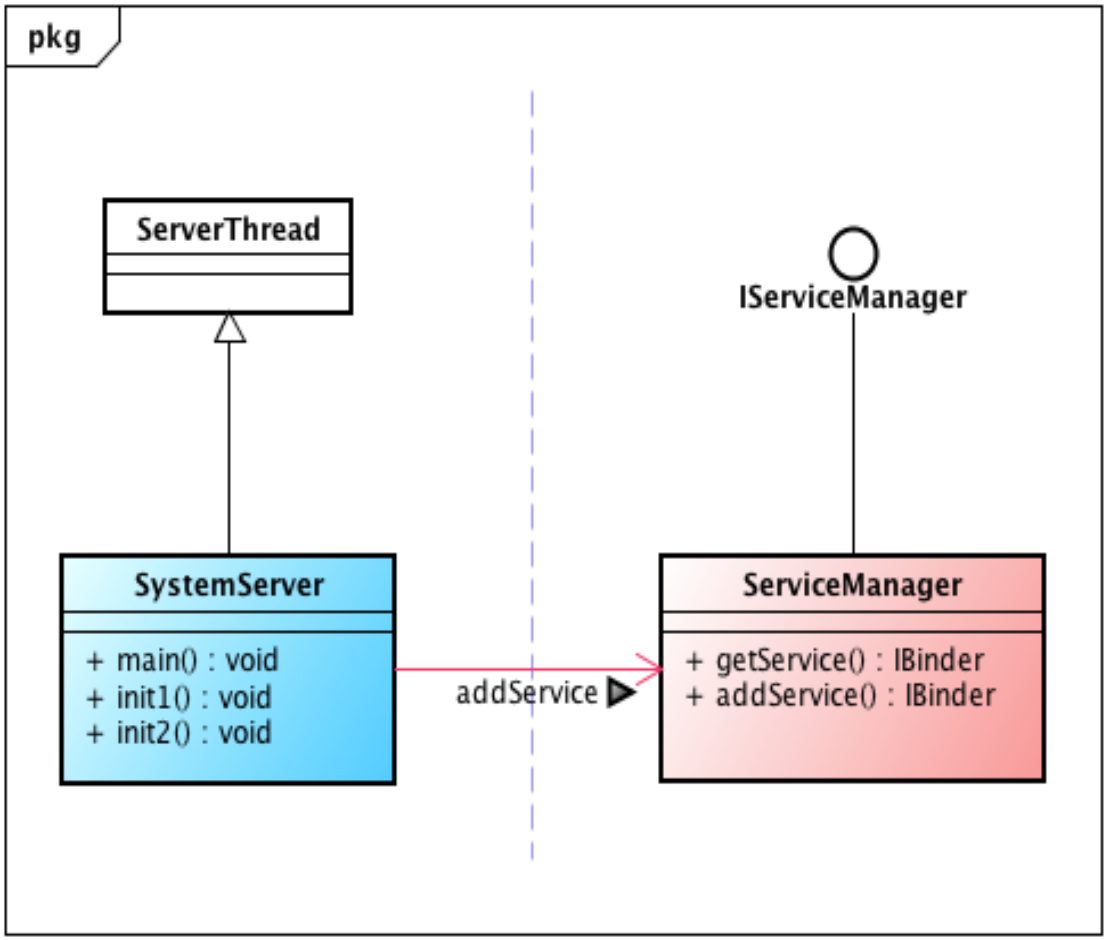
SurfaceFlinger Inner Architecture




ISurfaceComposerClient
ISurface
ISurfaceTexture
IGraphicBufferAlloc
ISurfaceComposer
这是什么小宇宙!?

Go……

Add Service


pkg
ServerThread
IServiceManager
SystemServer
+ main() : void
+ init1() : void
+ init2() : void
addService
ServiceManager
+ getService() : IBinder
+ addService() : IBinder

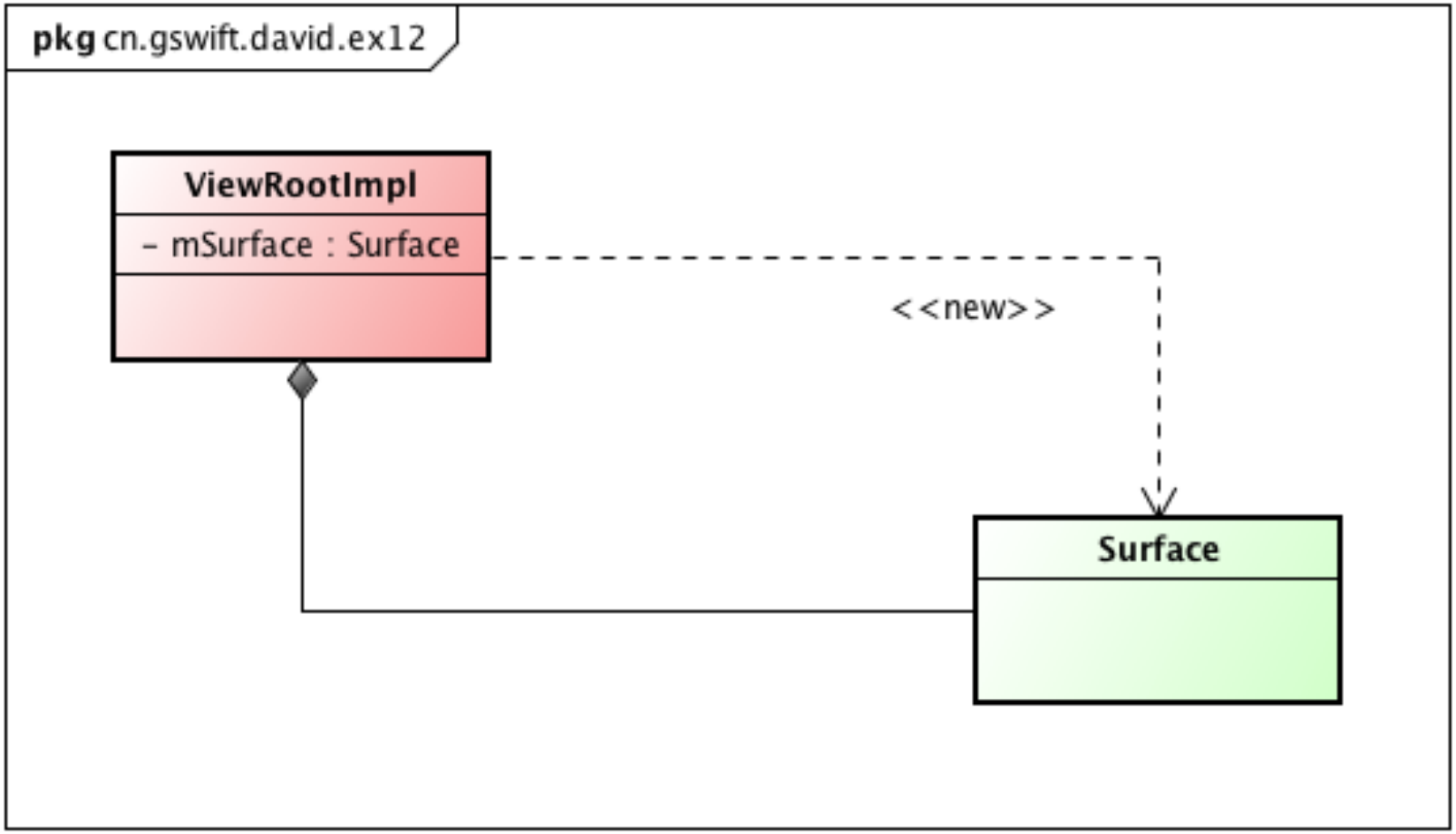
Executable SurfaceFlinger

1 #
main() –>
SurfaceFlinger::instantiate();–>
defaultServiceManager()->addService(String16(“SurfaceFlinger”), new SurfaceFlinger());
2 #
main() –>
system_init(); –>
SurfaceFlinger::instantiate();

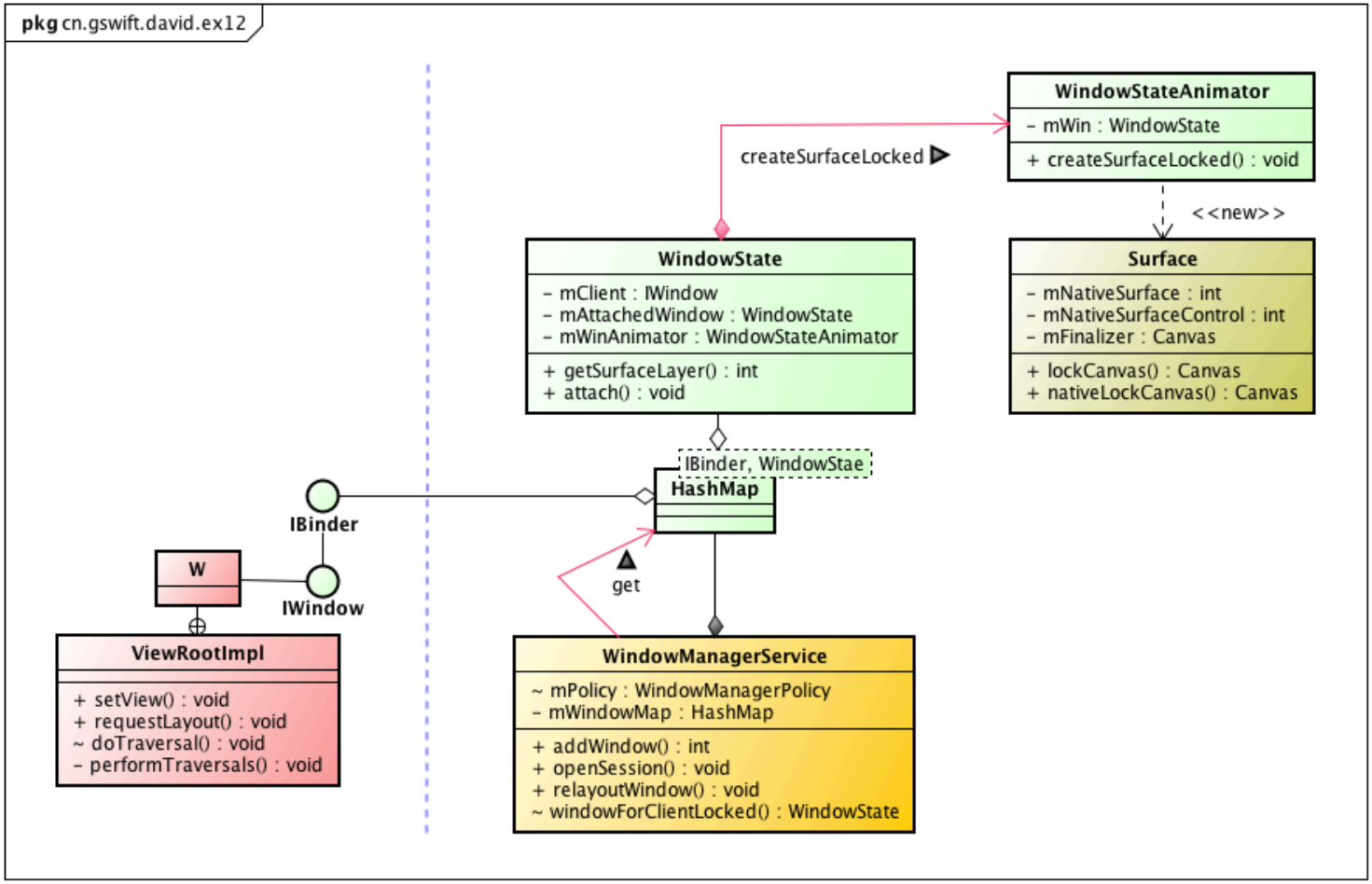
Surface

SurfaceFlinger

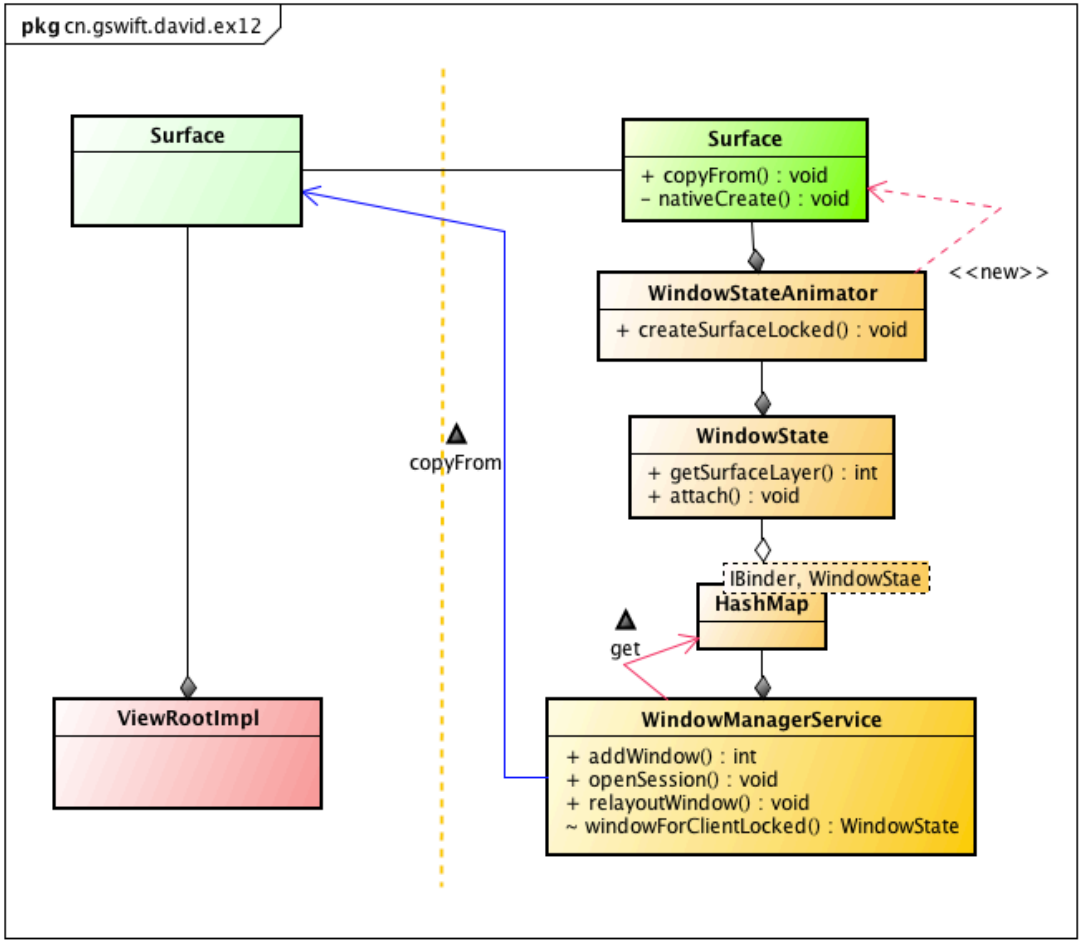
ViewRootImpl has a Surface



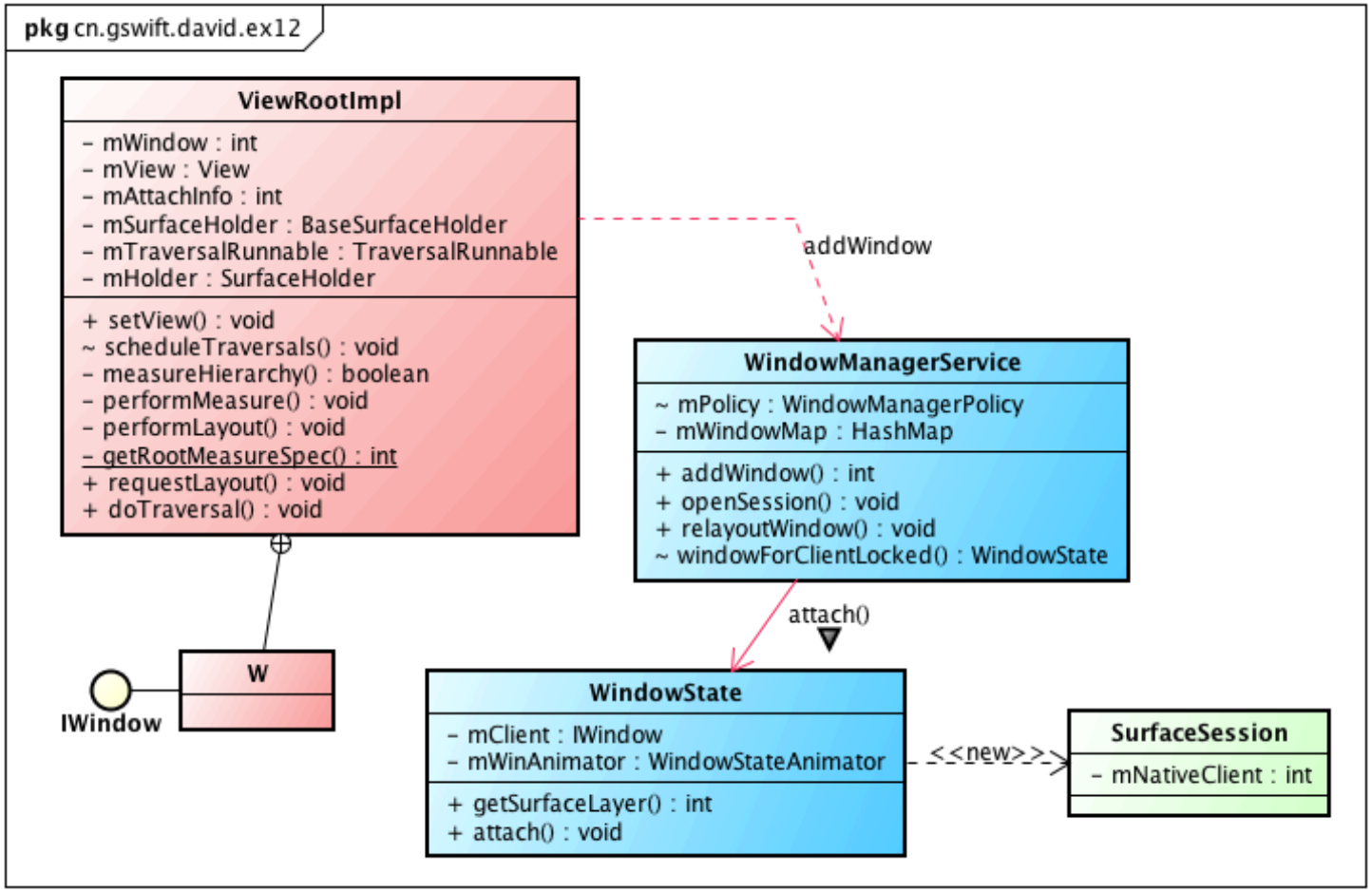
Create a surface (java)



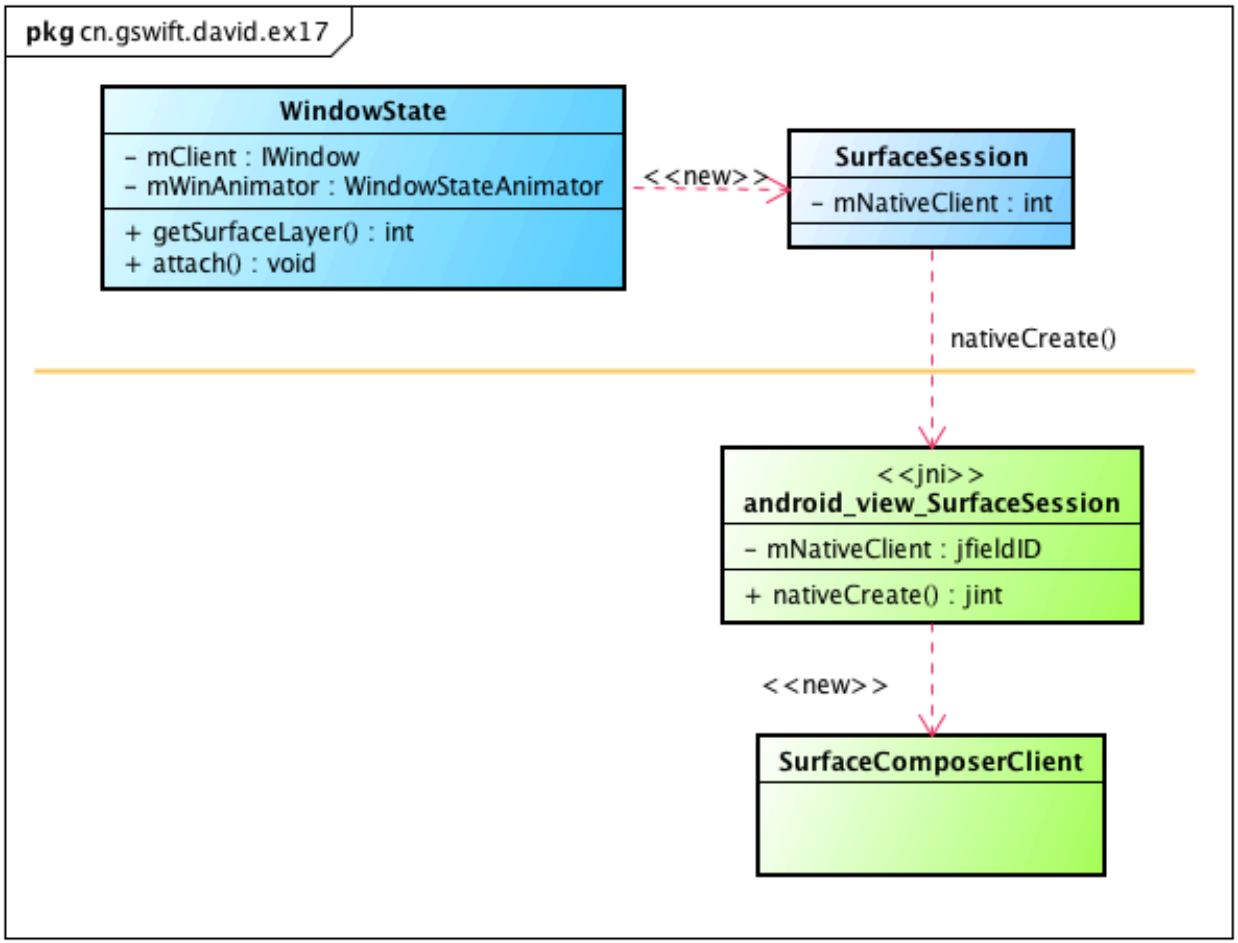
Send back the newly created surface object to ViewRootImpl as a parcel



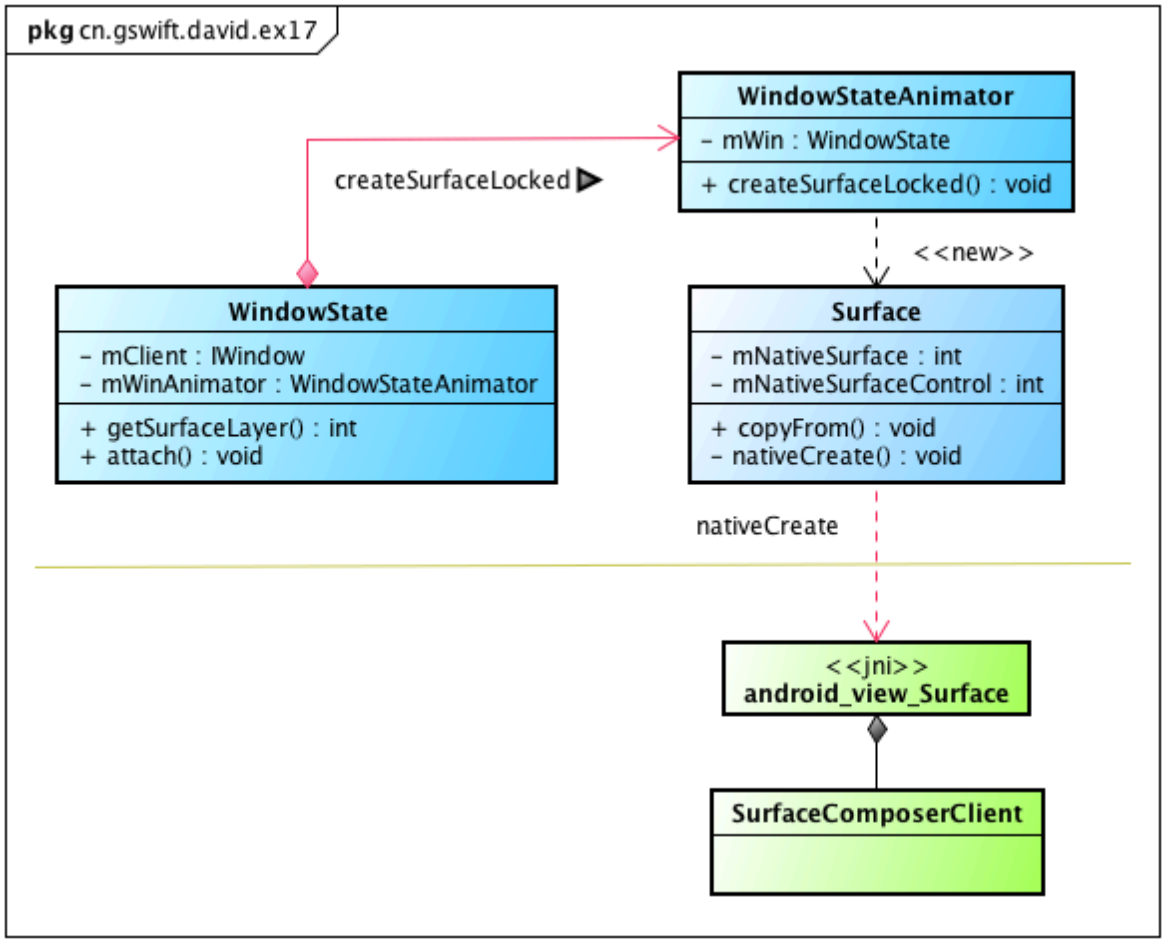
Create a SurfaceSession(Java)



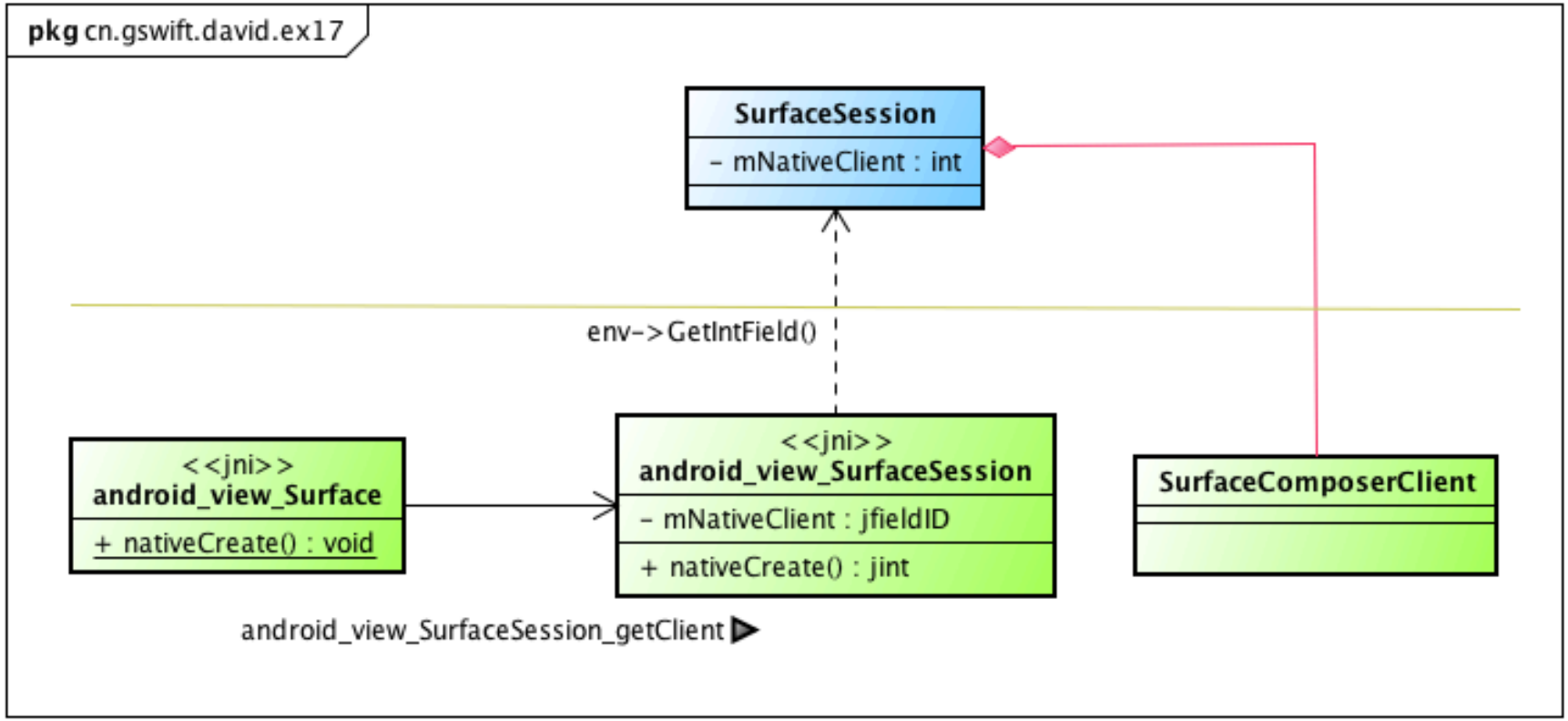
Create a native SurfaceComposerClient



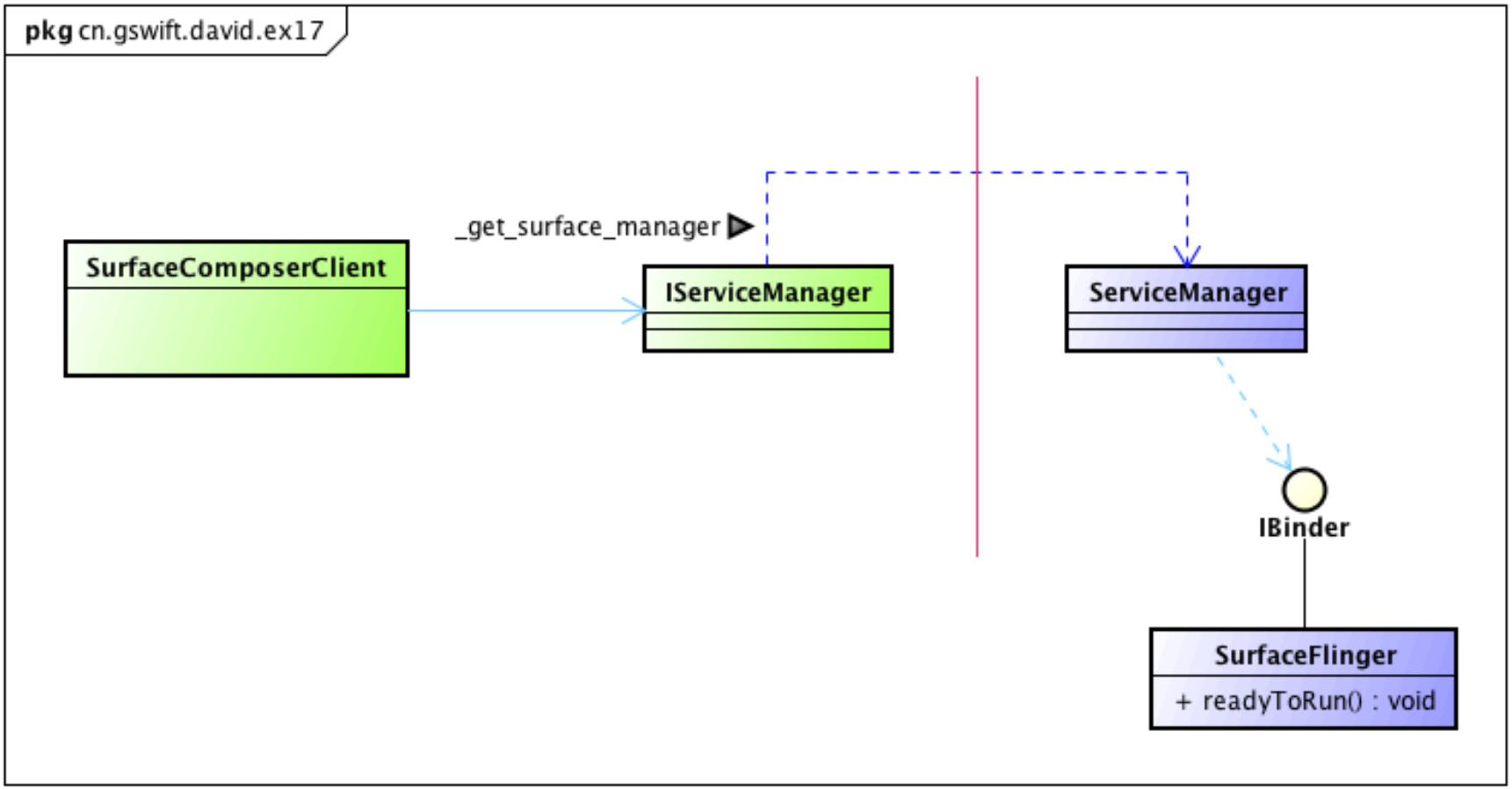
Send back the SurfaceComposerClient's reference (C++) to java's Surface



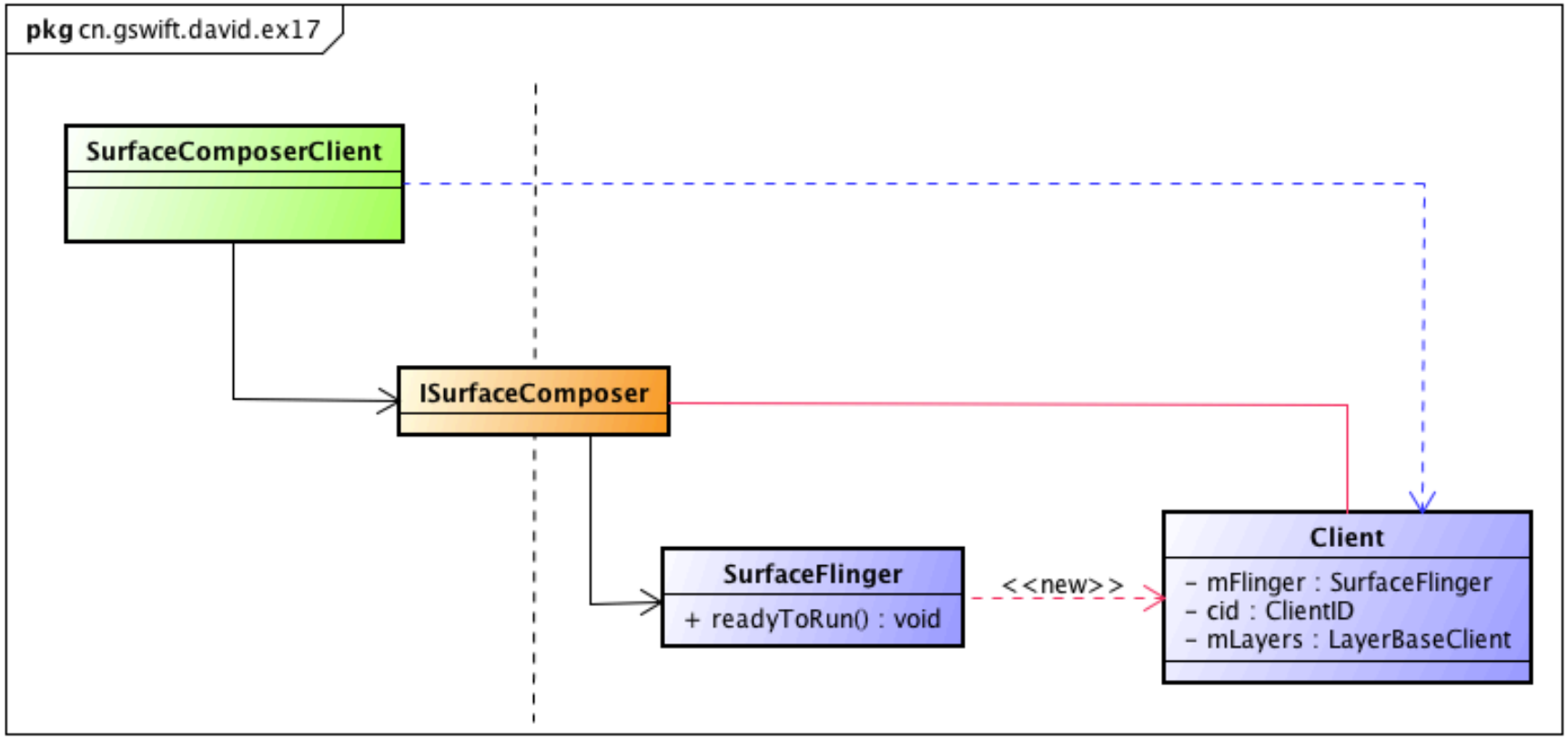
Get the SurfaceComposerClient's reference



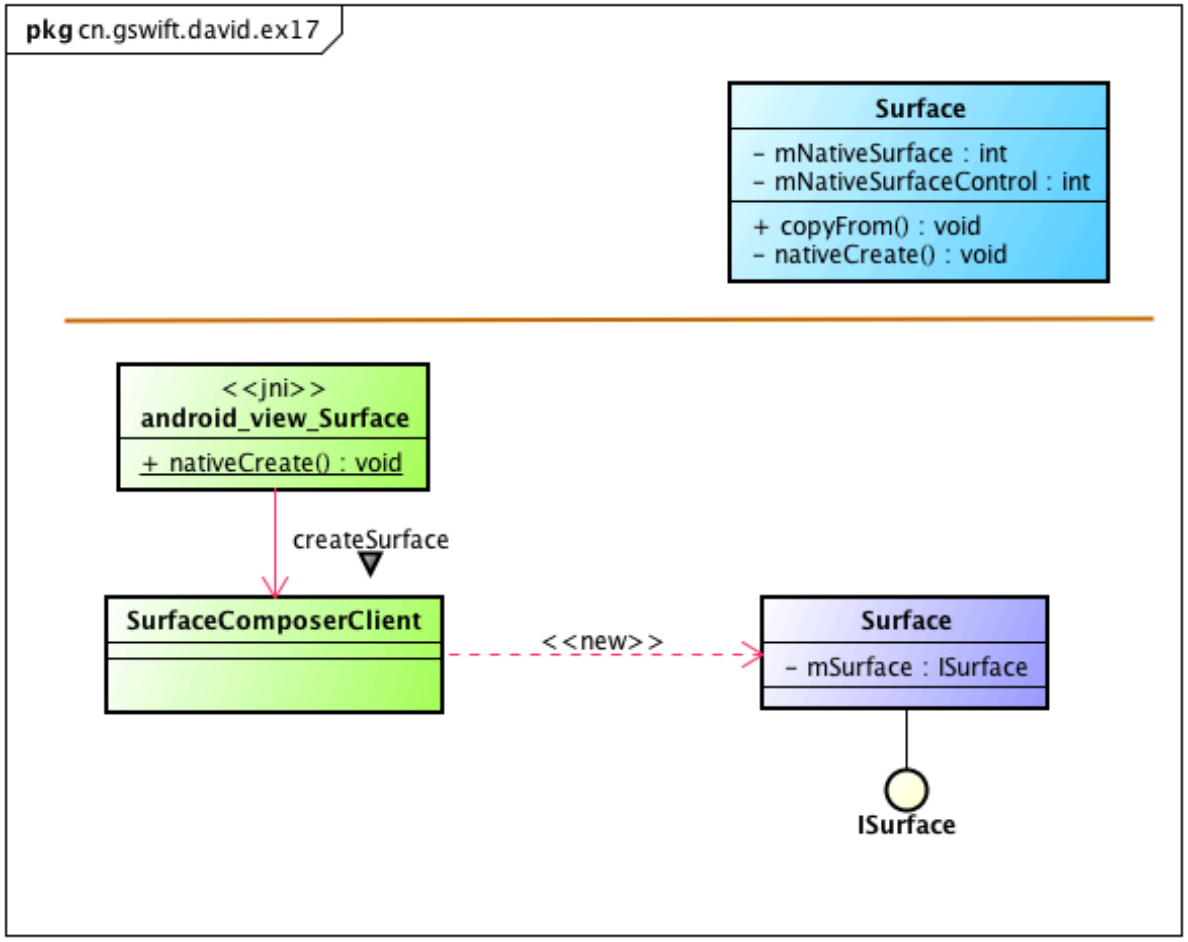
Get a SurfaceFlinger interface



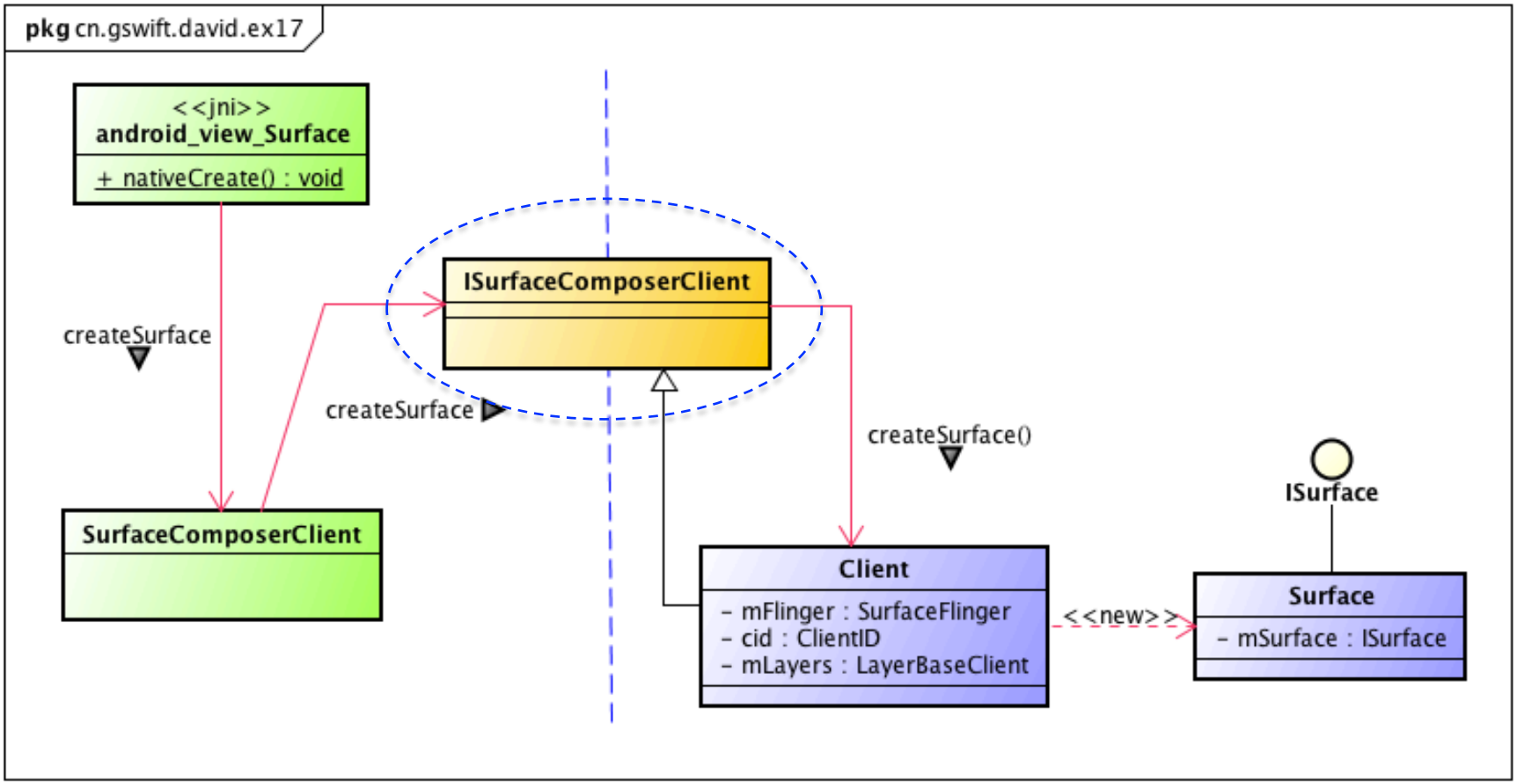
SurfaceComposerClient and its' Client in SurfaceFlinger



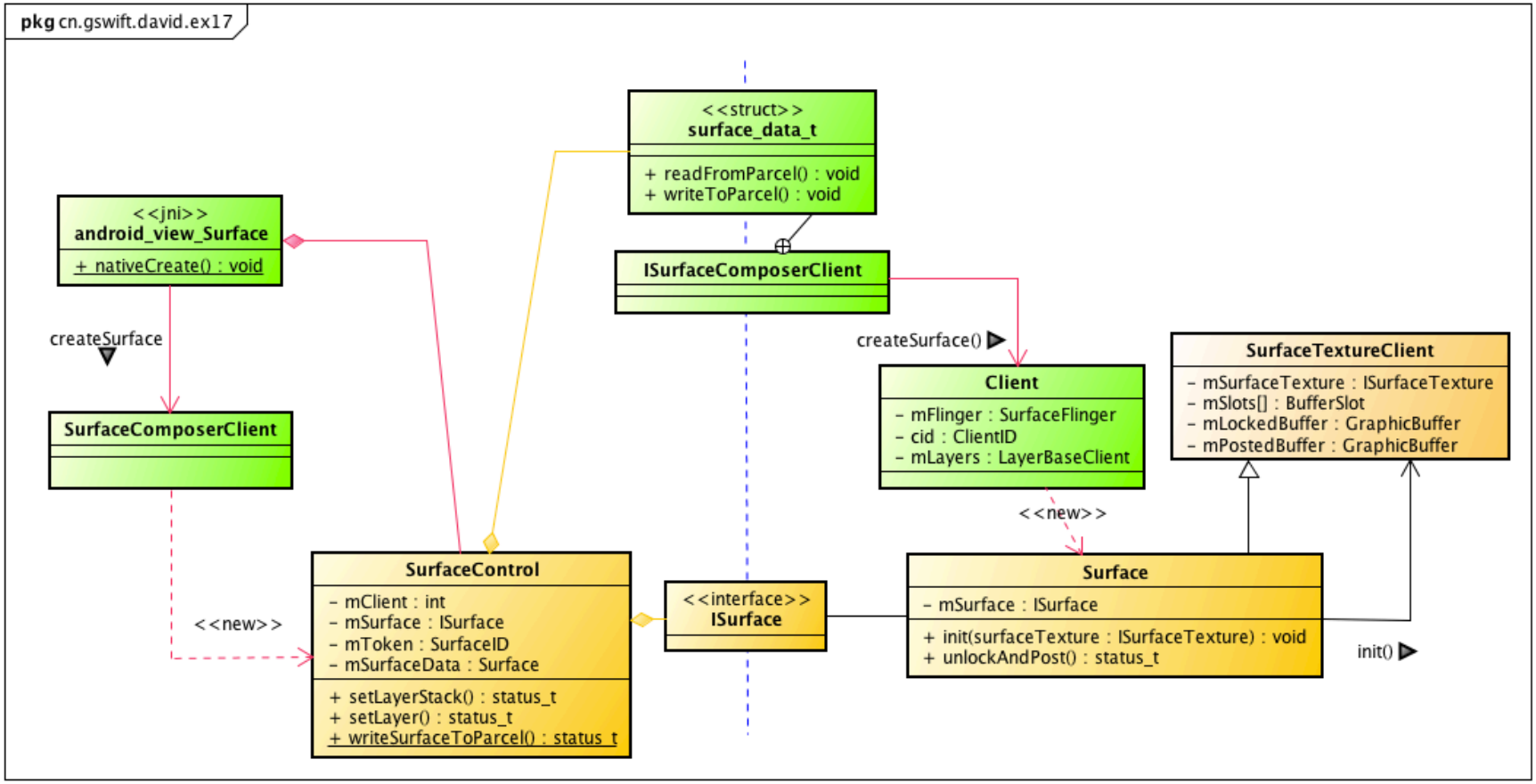
Create a native Surface in SurfaceFlinger (1)



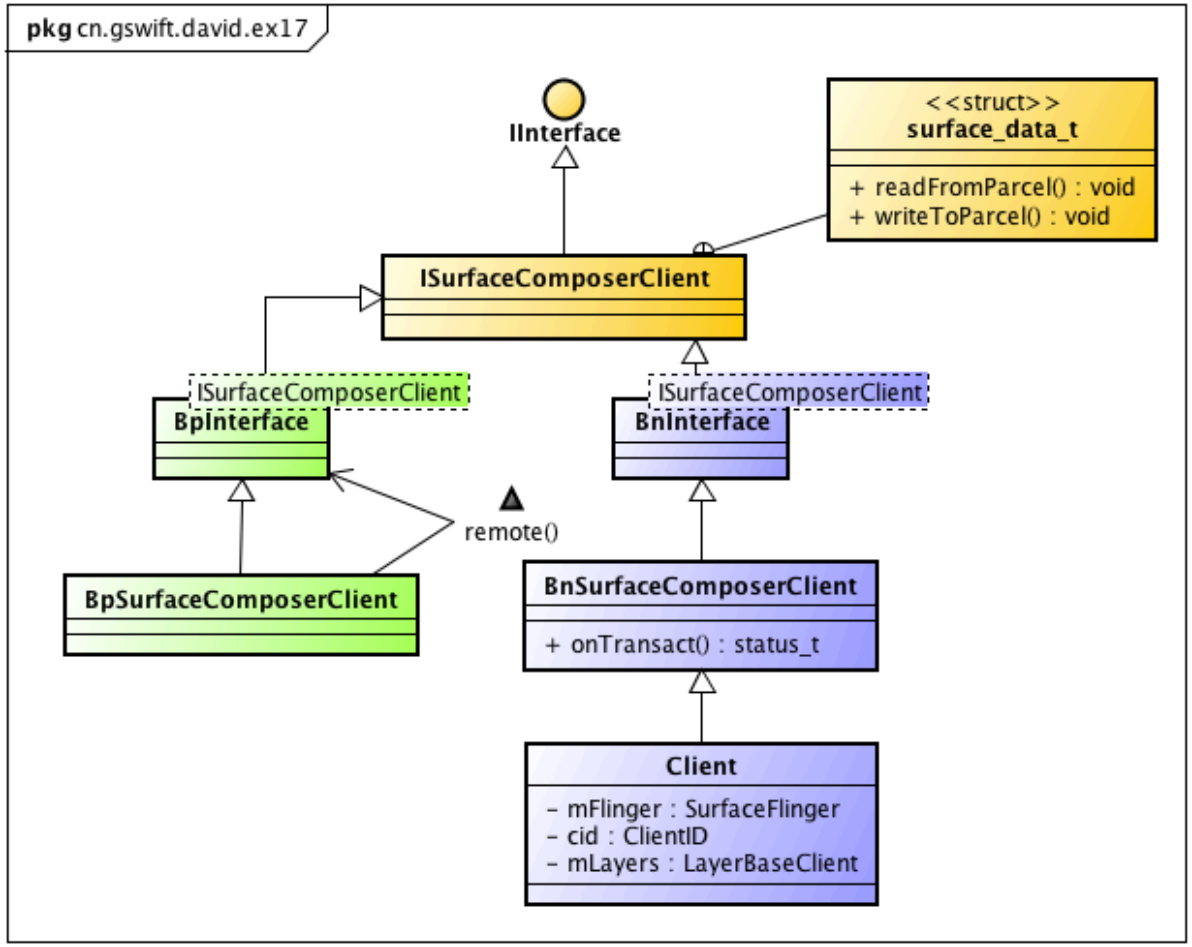
Create a native Surface in SurfaceFlinger (2)



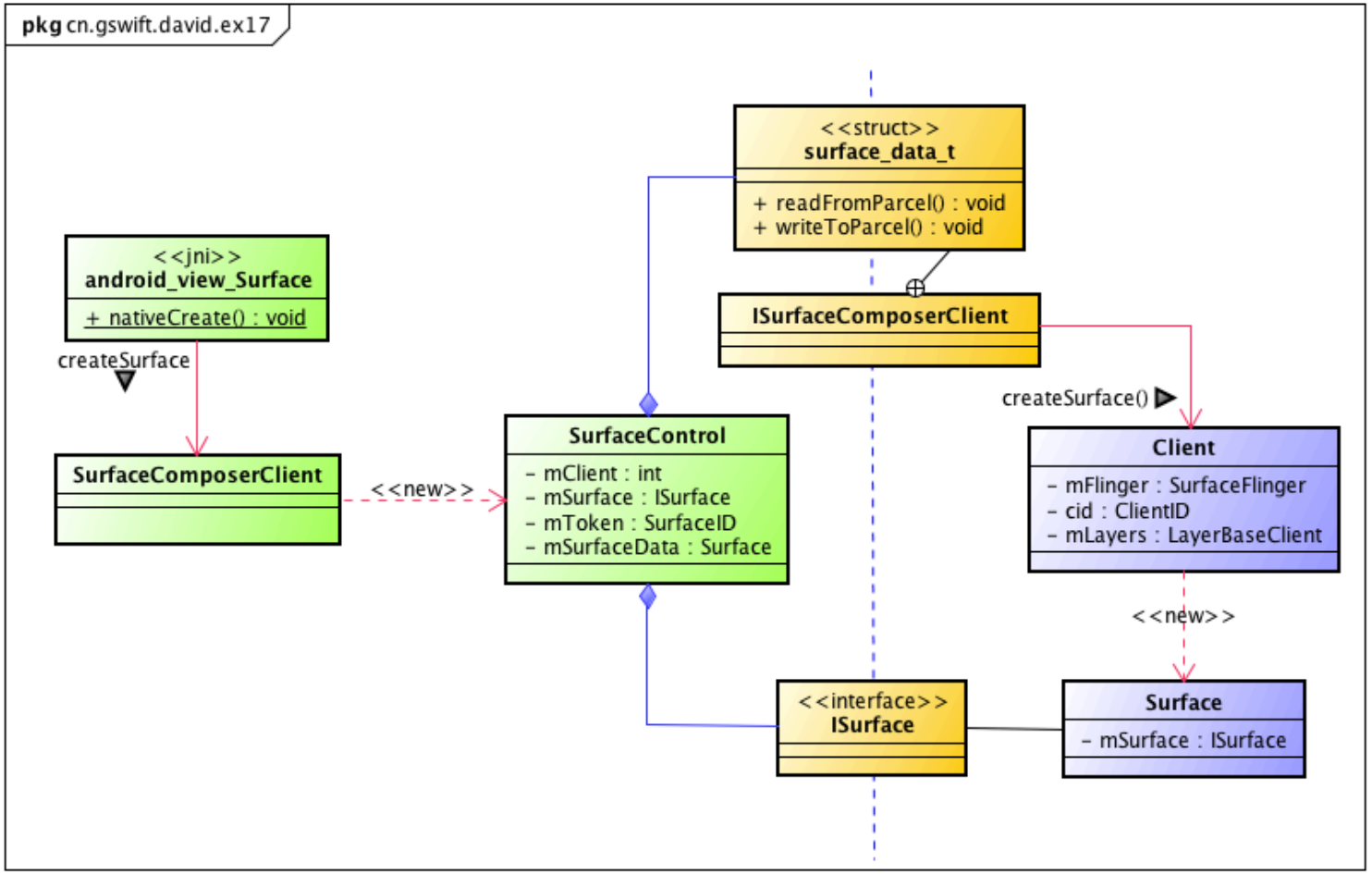
Create a native Surface in SurfaceFlinger (3)



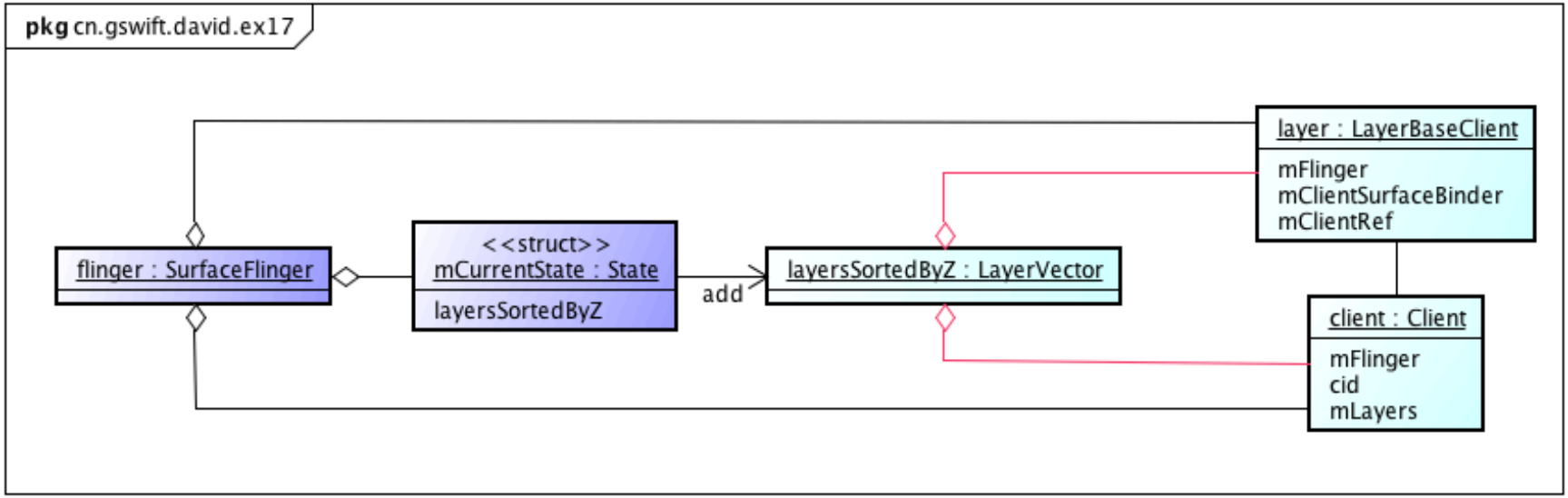
ISurfaceComposerClient



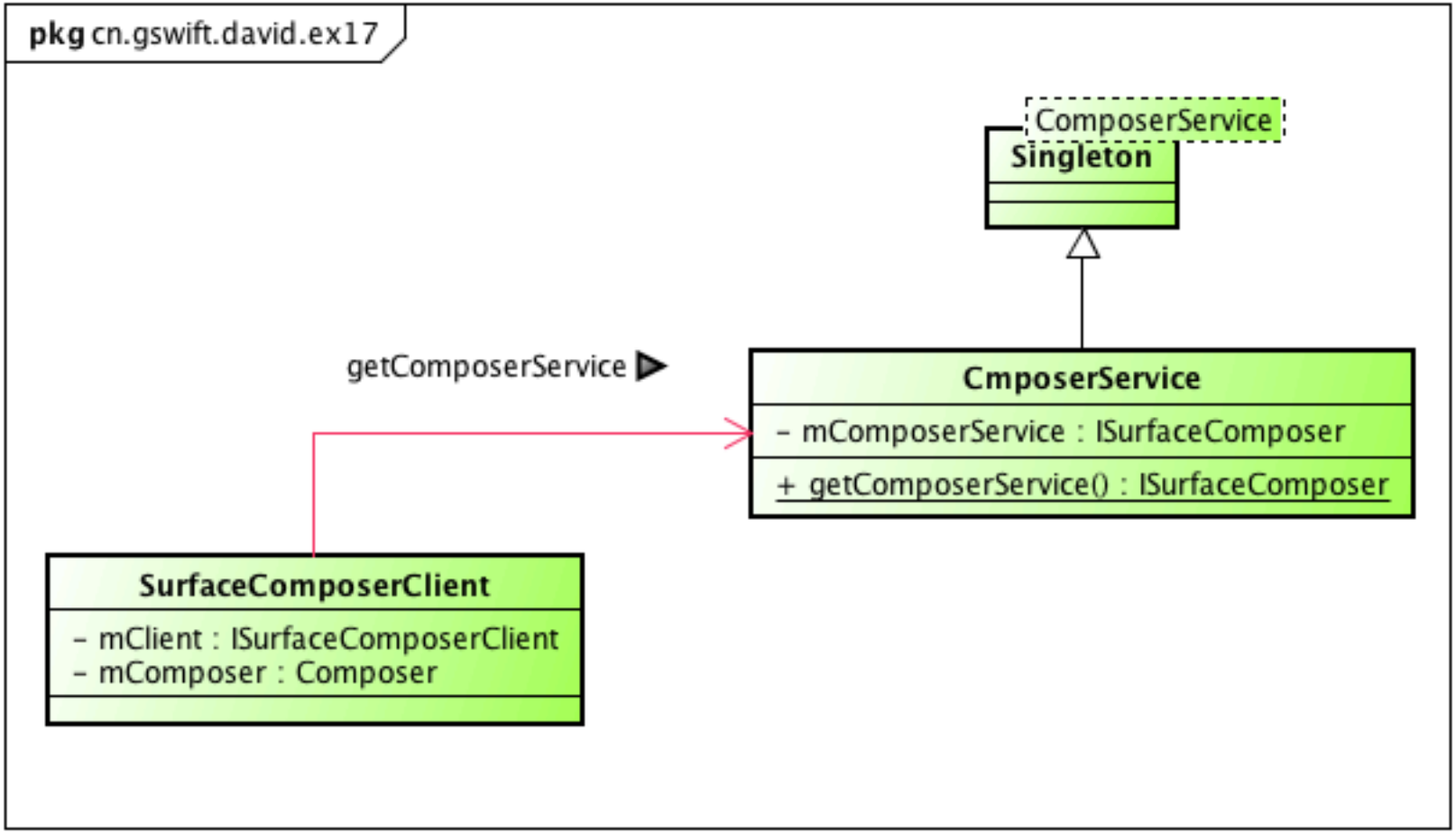
Create a SurfaceControl



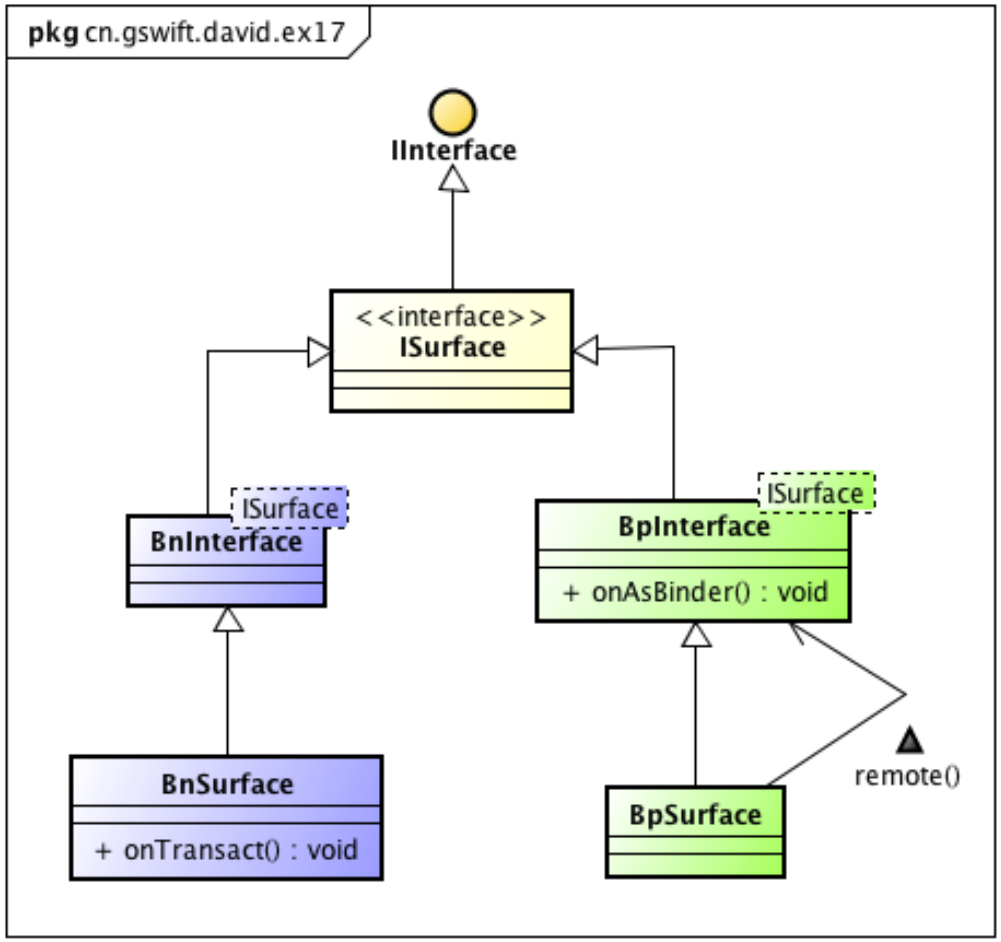
Objects in SurfaceFlinger



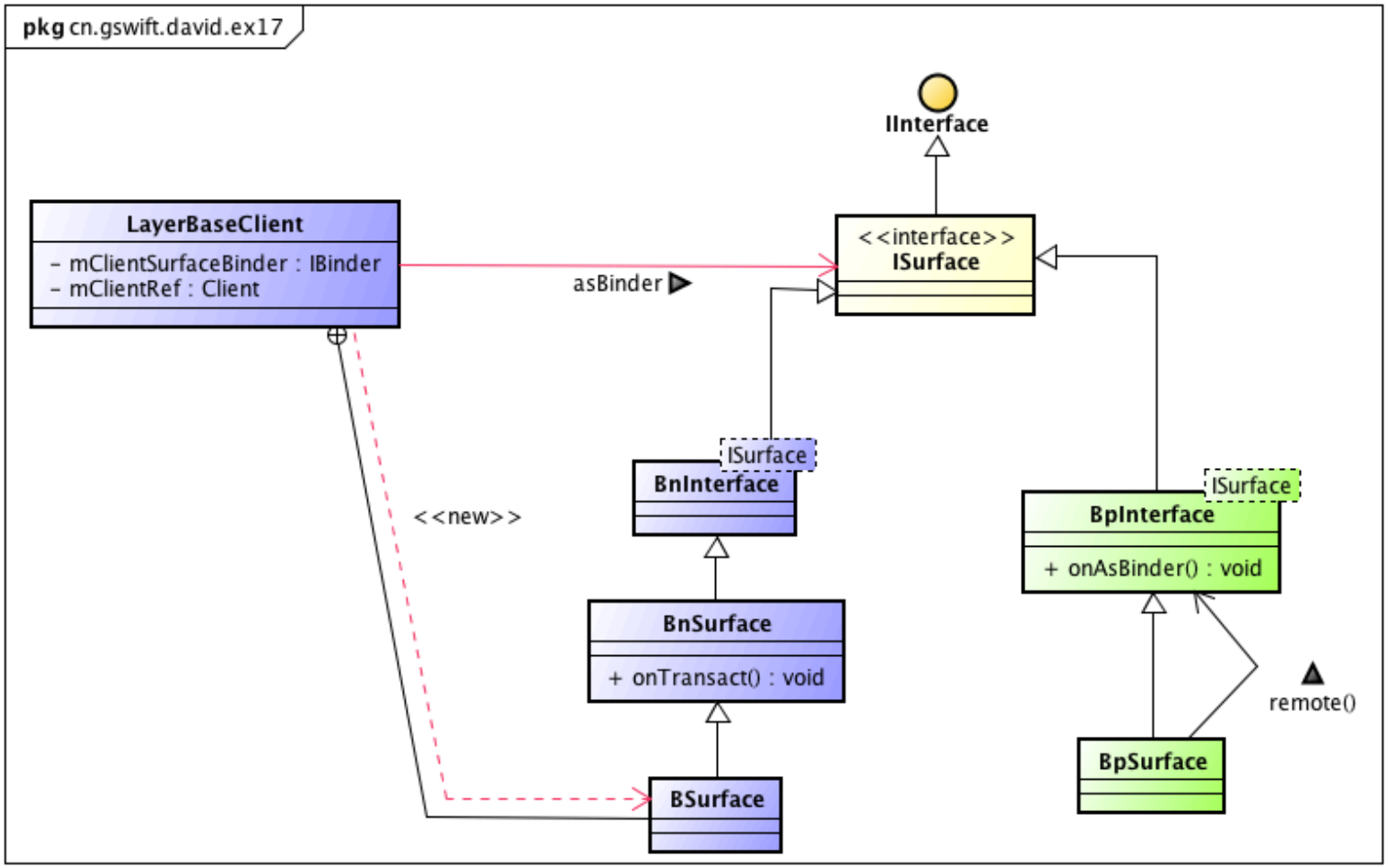
Get the ComposerService



ISurface

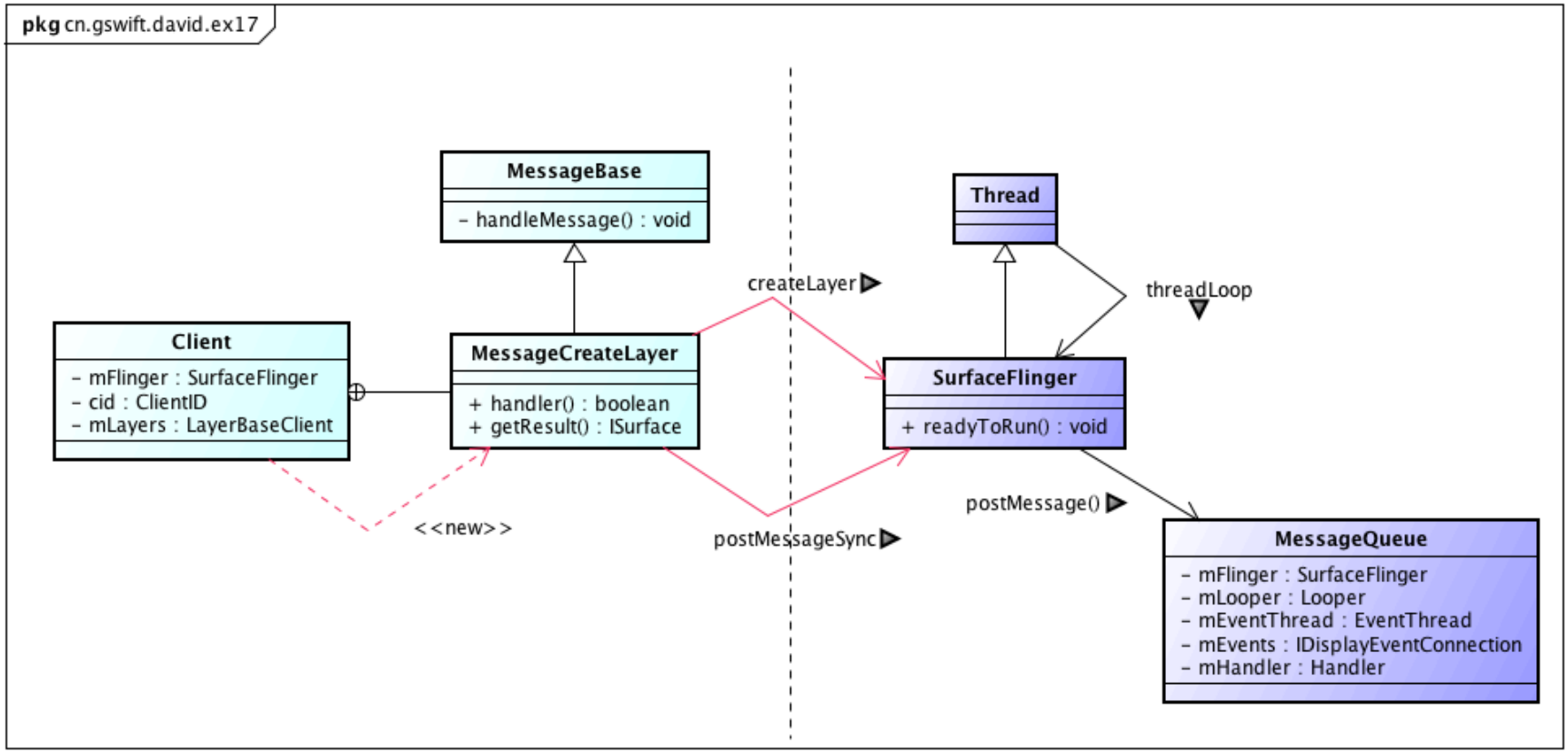

pkg cn.gswift.david.ex17
IInterface
<<interface>>
ISurface
ISurface
BnInterface
ISurface
BpInterface
+ onAsBinder() : void
BnSurface
+ onTransact() : void
BpSurface
remote()

Create a BSurface

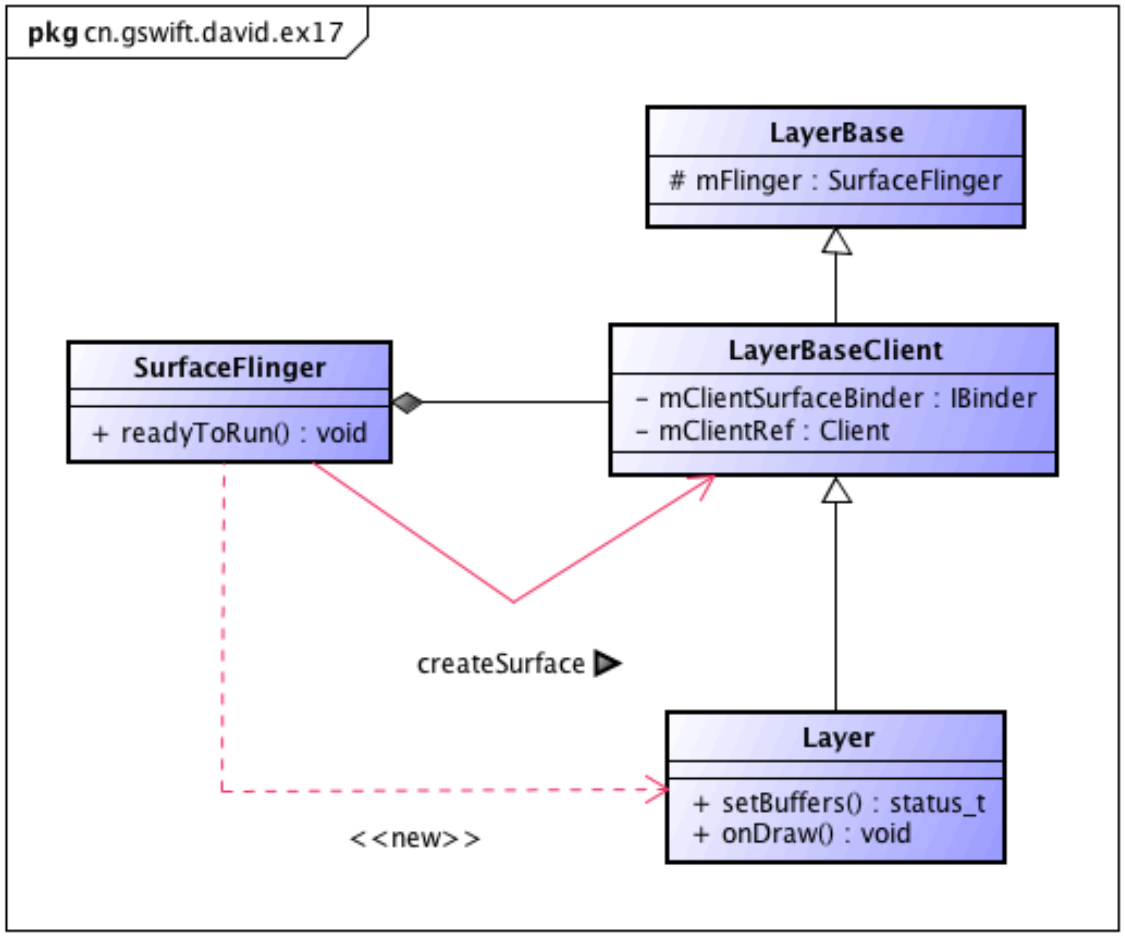


SurfaceFlinger Layer

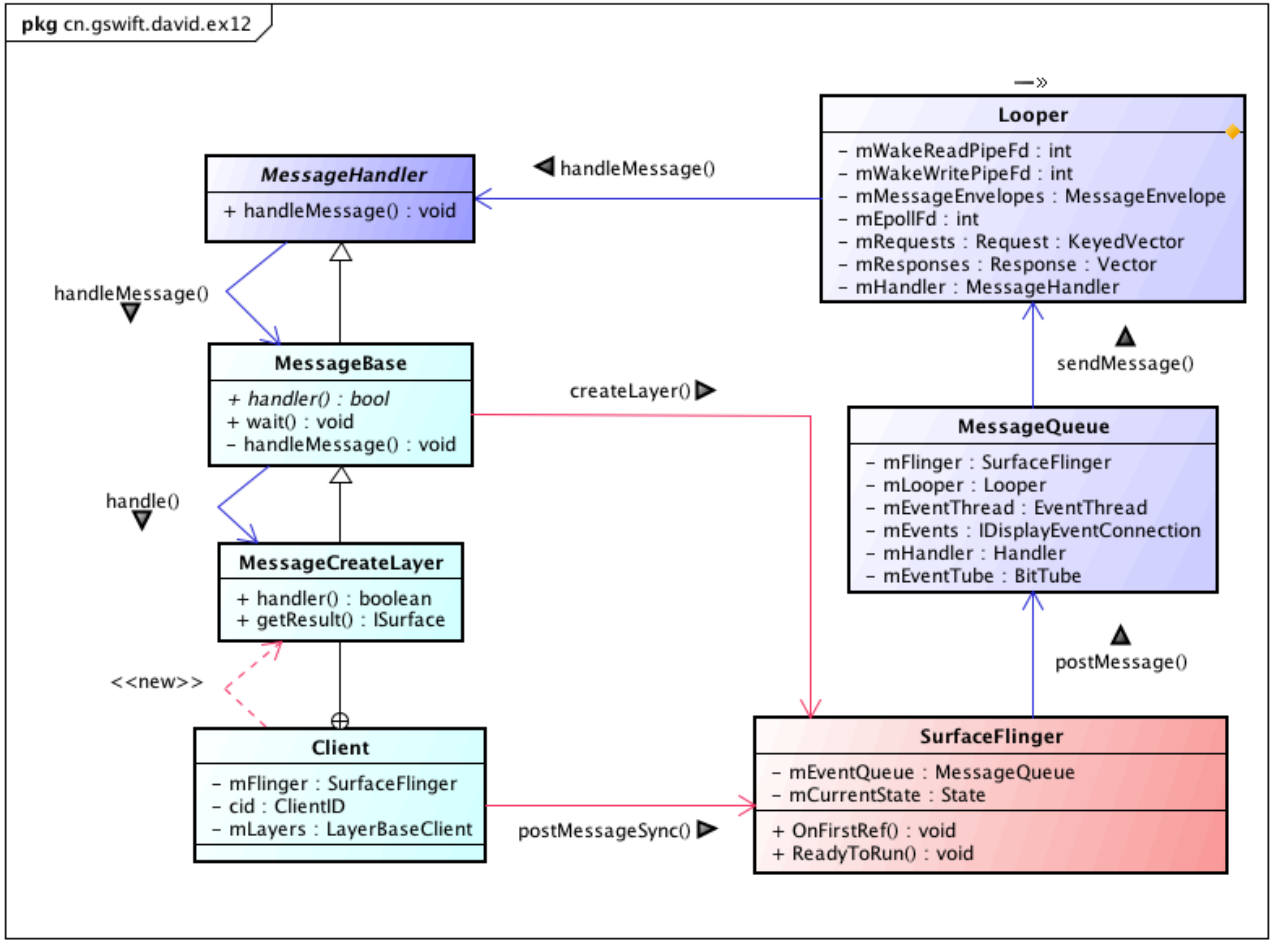
Create a Layer by Client



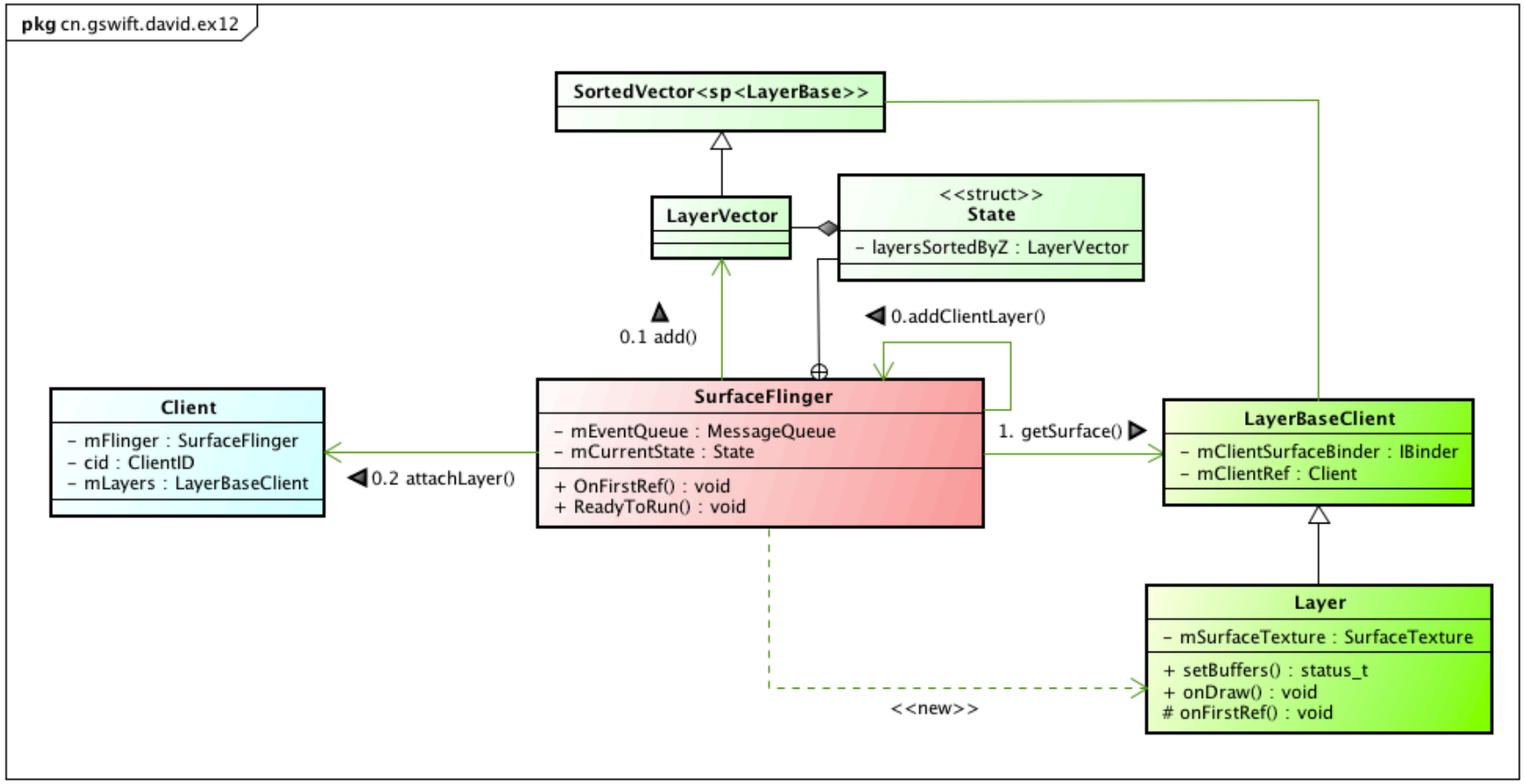
Create a Layer in SurfaceFlinger (1)

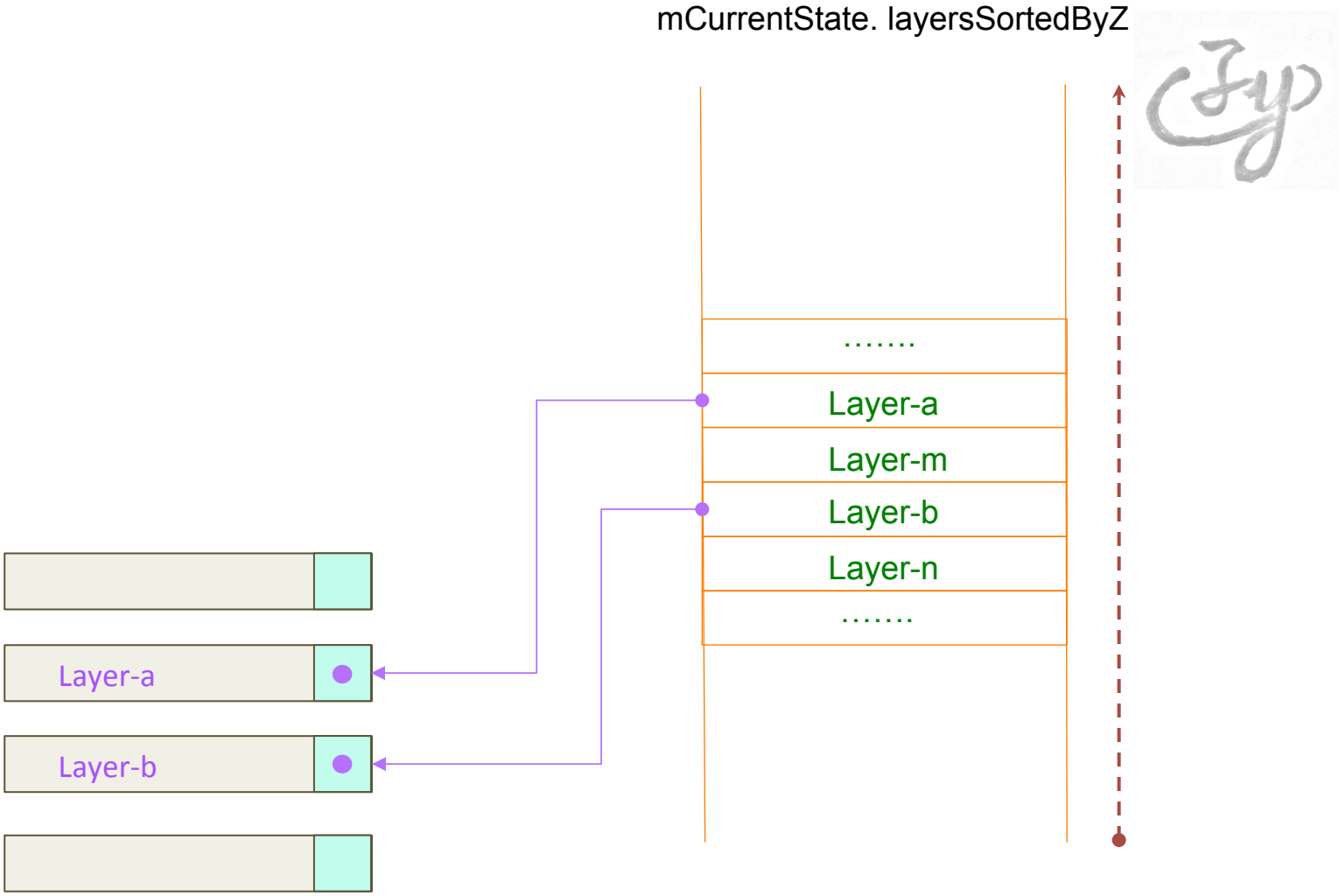


Create a Layer in SurfaceFlinger (2)



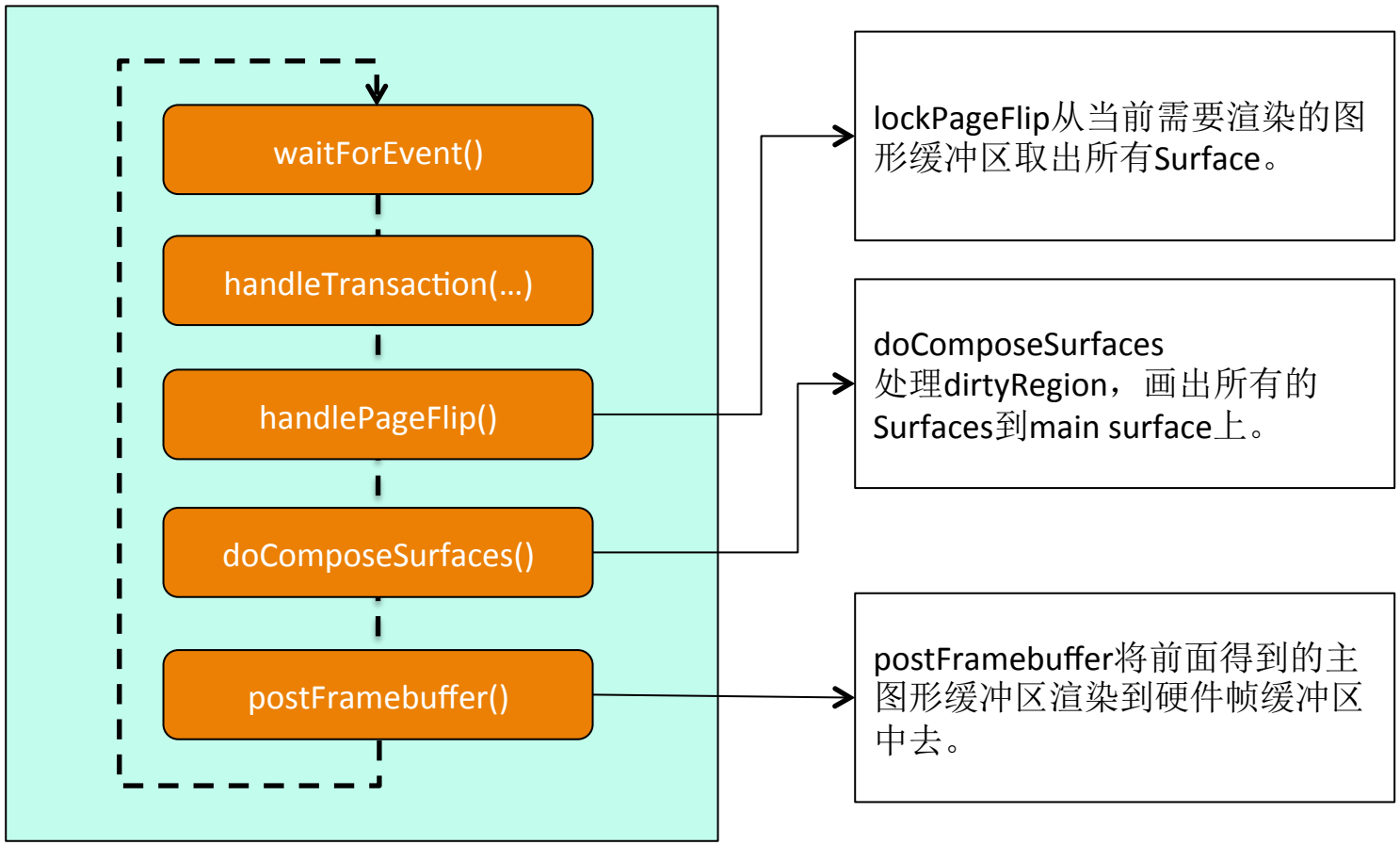
Create a Layer in SurfaceFlinger(3)




mCurrentState. layersSortedByZ
.....
Layer-a
Layer-m
Layer-b
Layer-n
.....
Layer-a
Layer-b

Surface Flinger

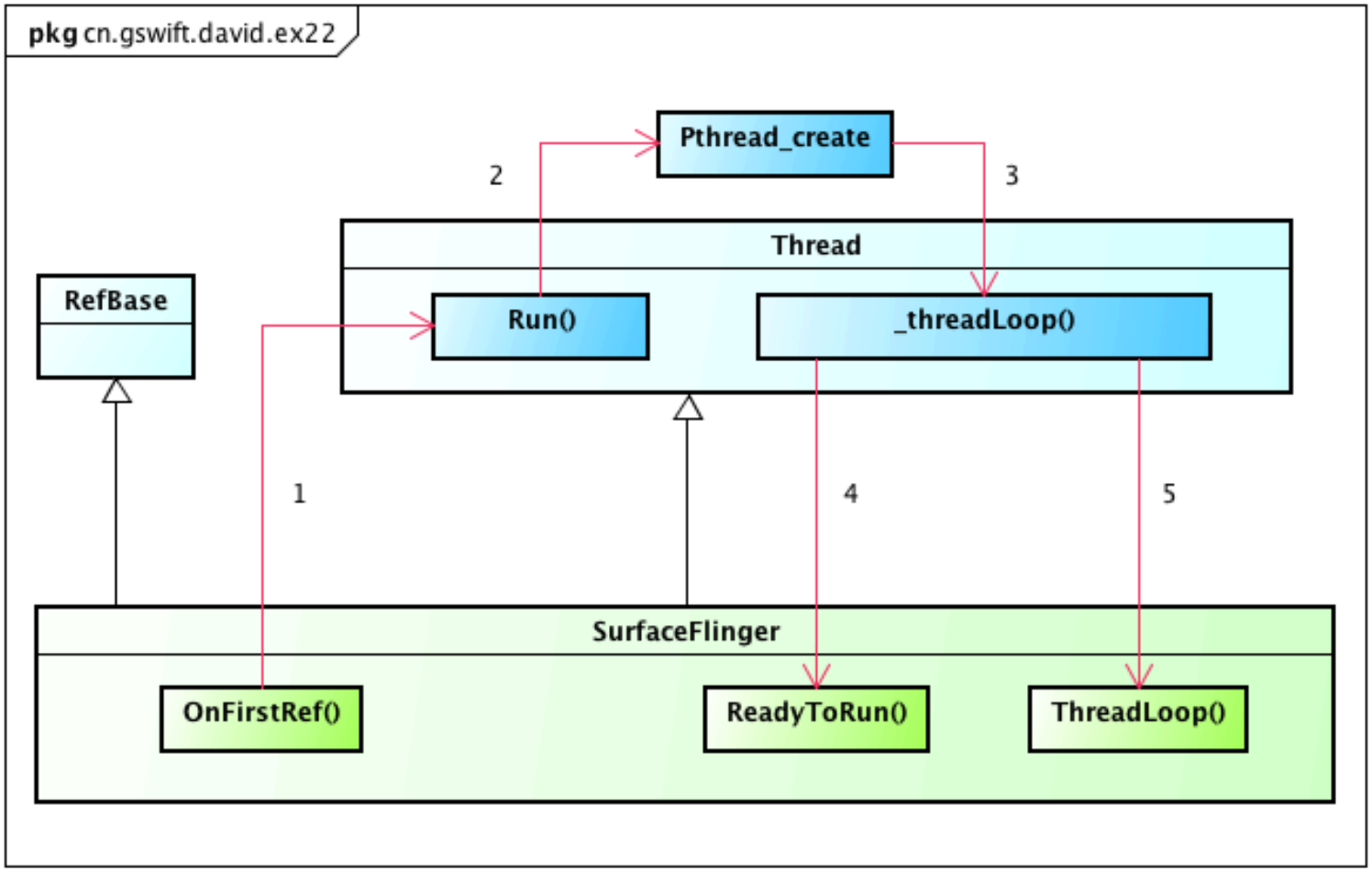
SurfaceFlinger::threadLoop()



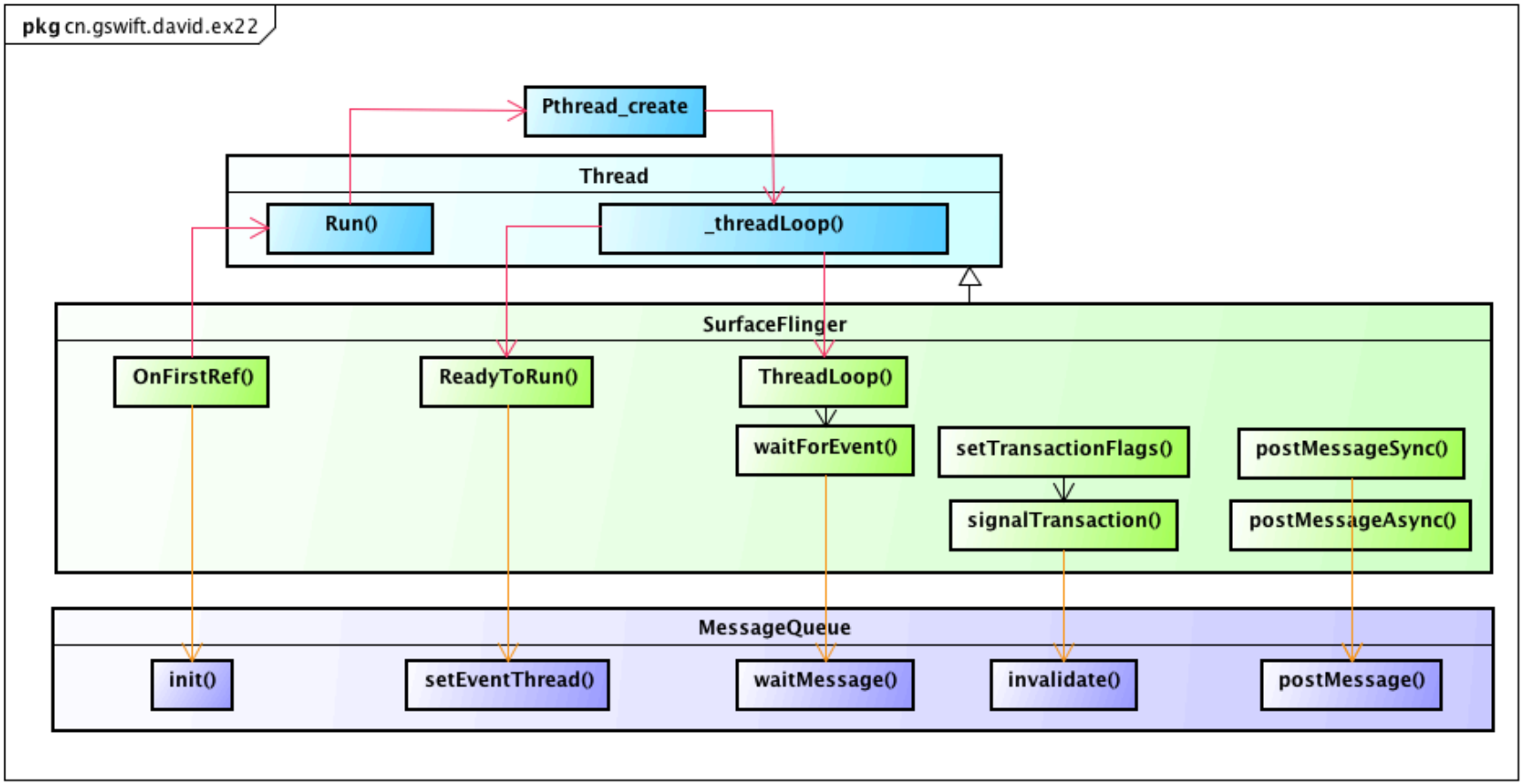
WaitForEvents

SURFACEFLINGER

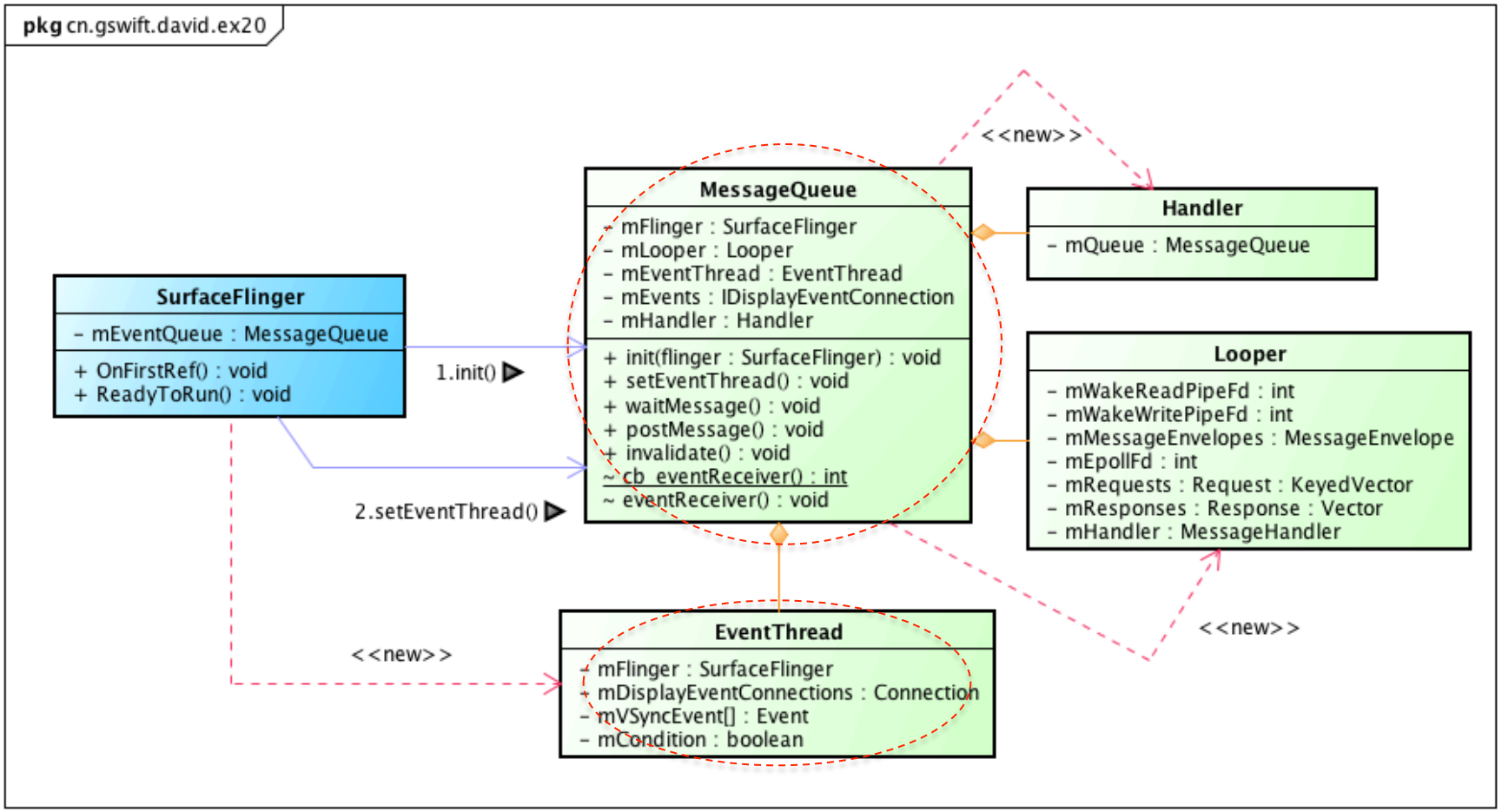
SurfaceFlinger Thread



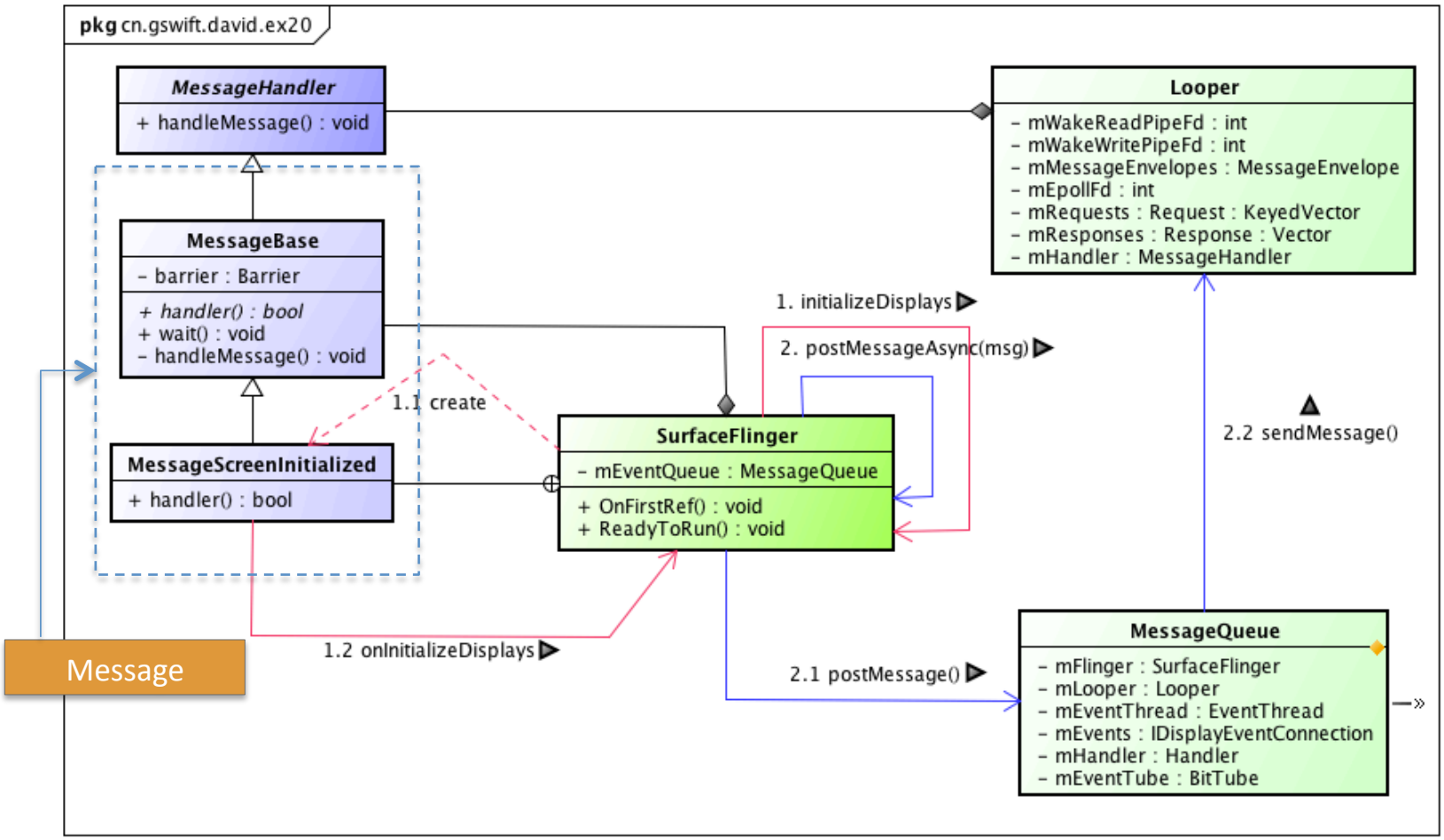
SurfaceFlinger Thread & MessageQueue



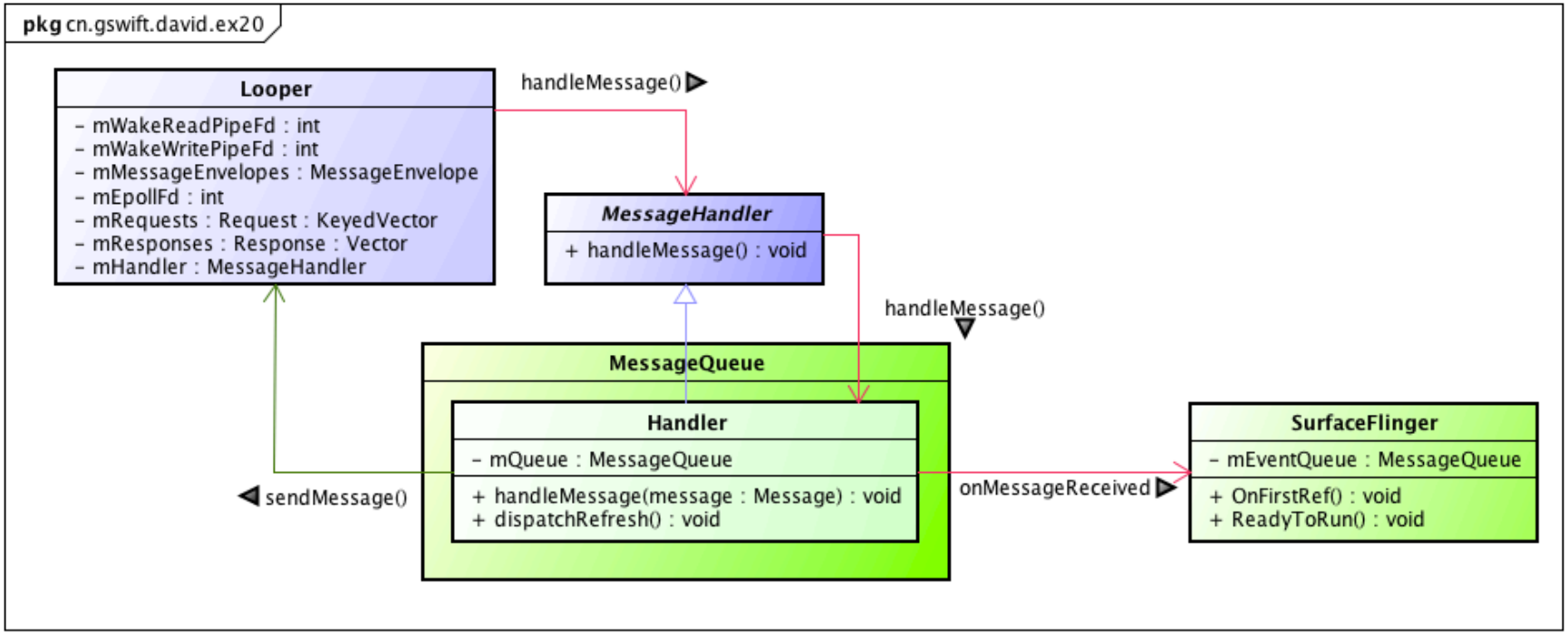
EventThread/ Messagequeue/Hanlder/Looper



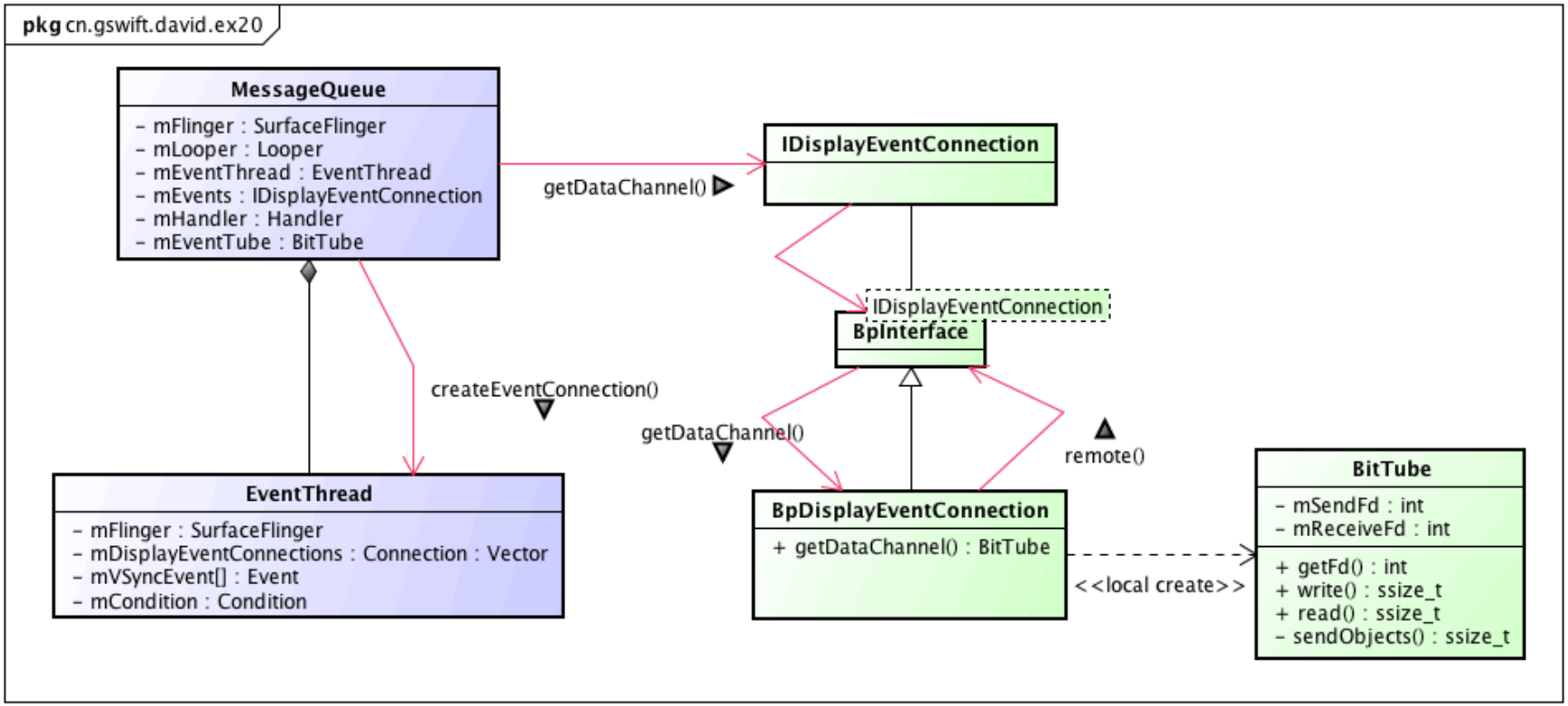
SurfaceFlinger postMessageAsync()



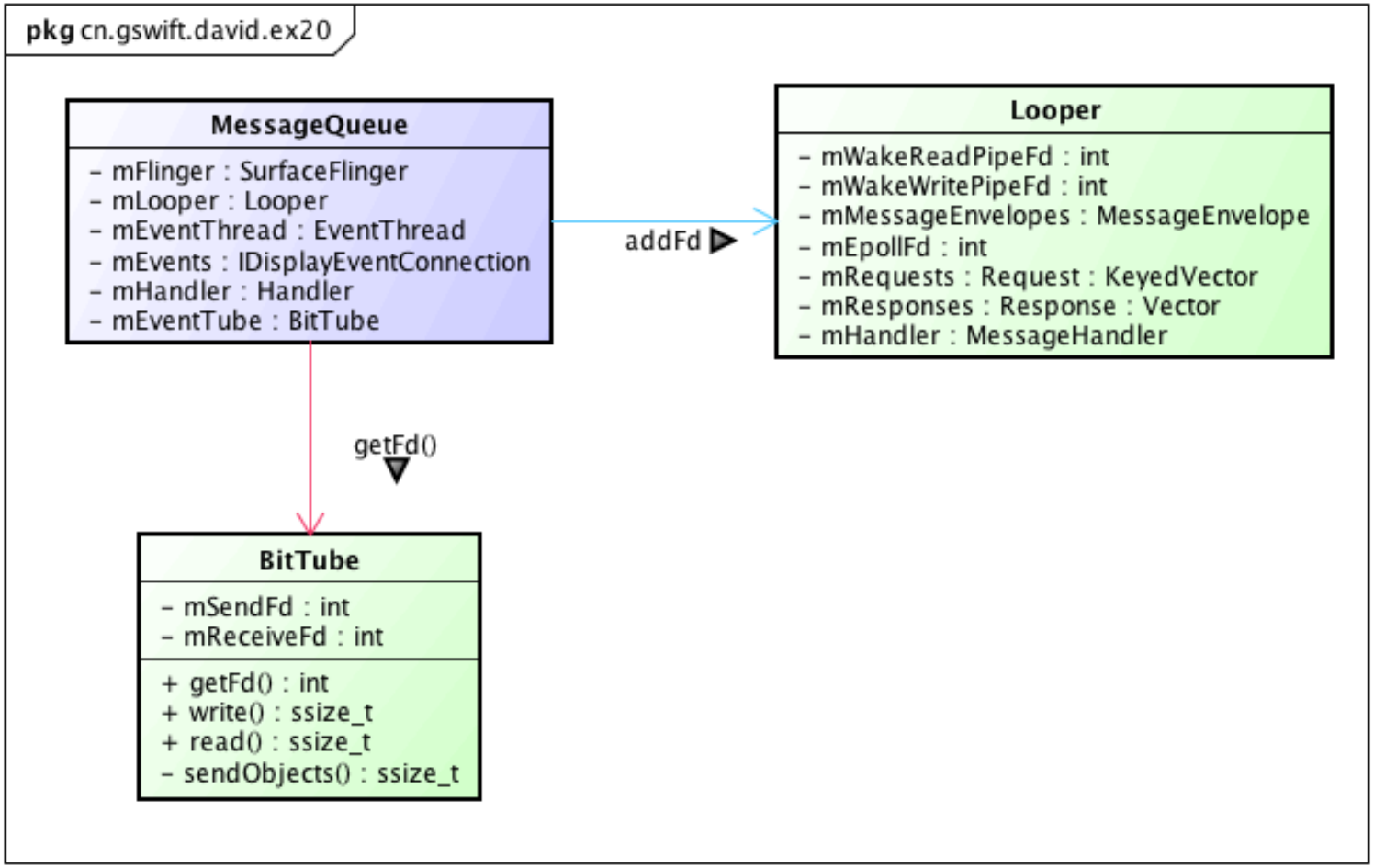
handleMessage



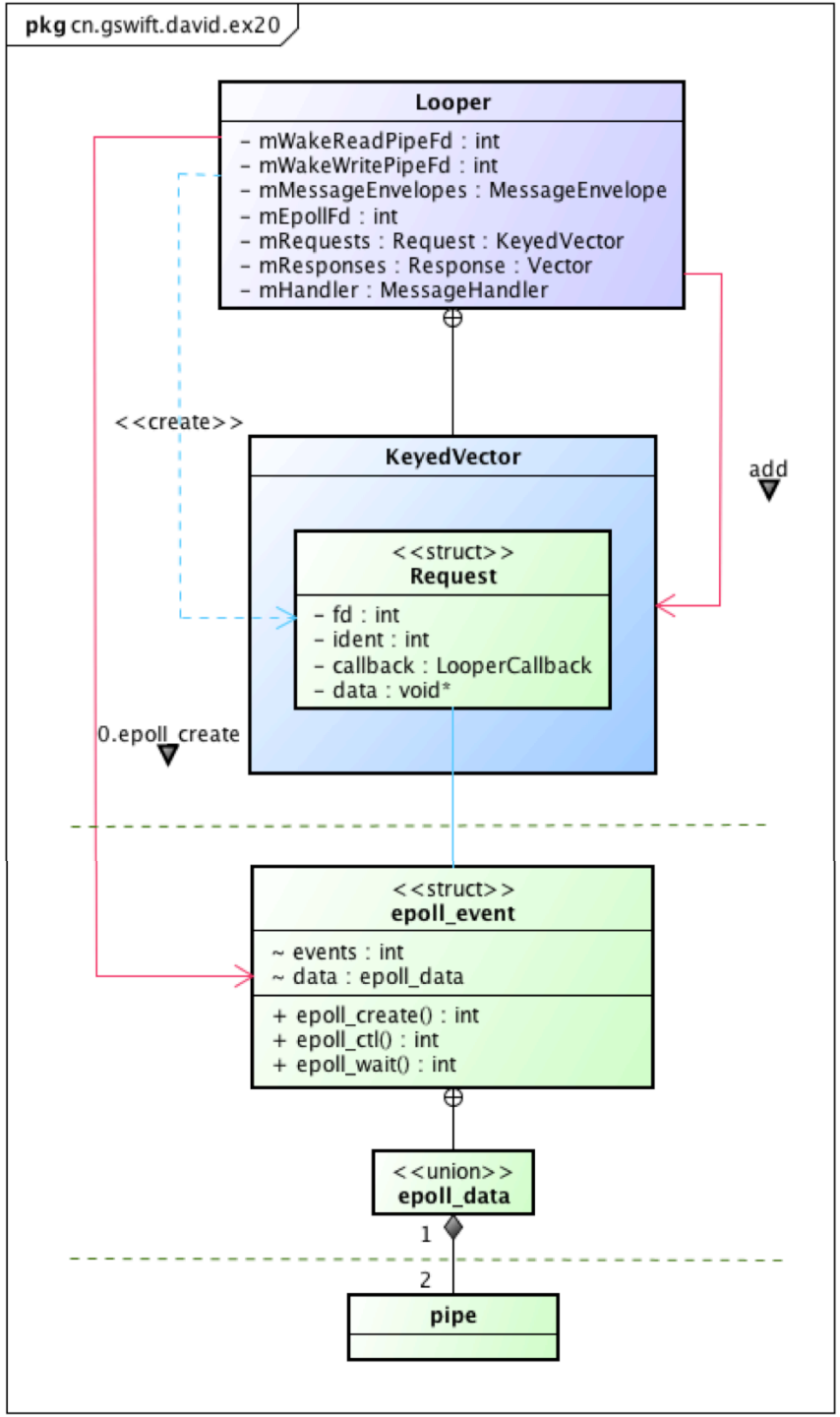
MessageQueue::setEventThread<1>



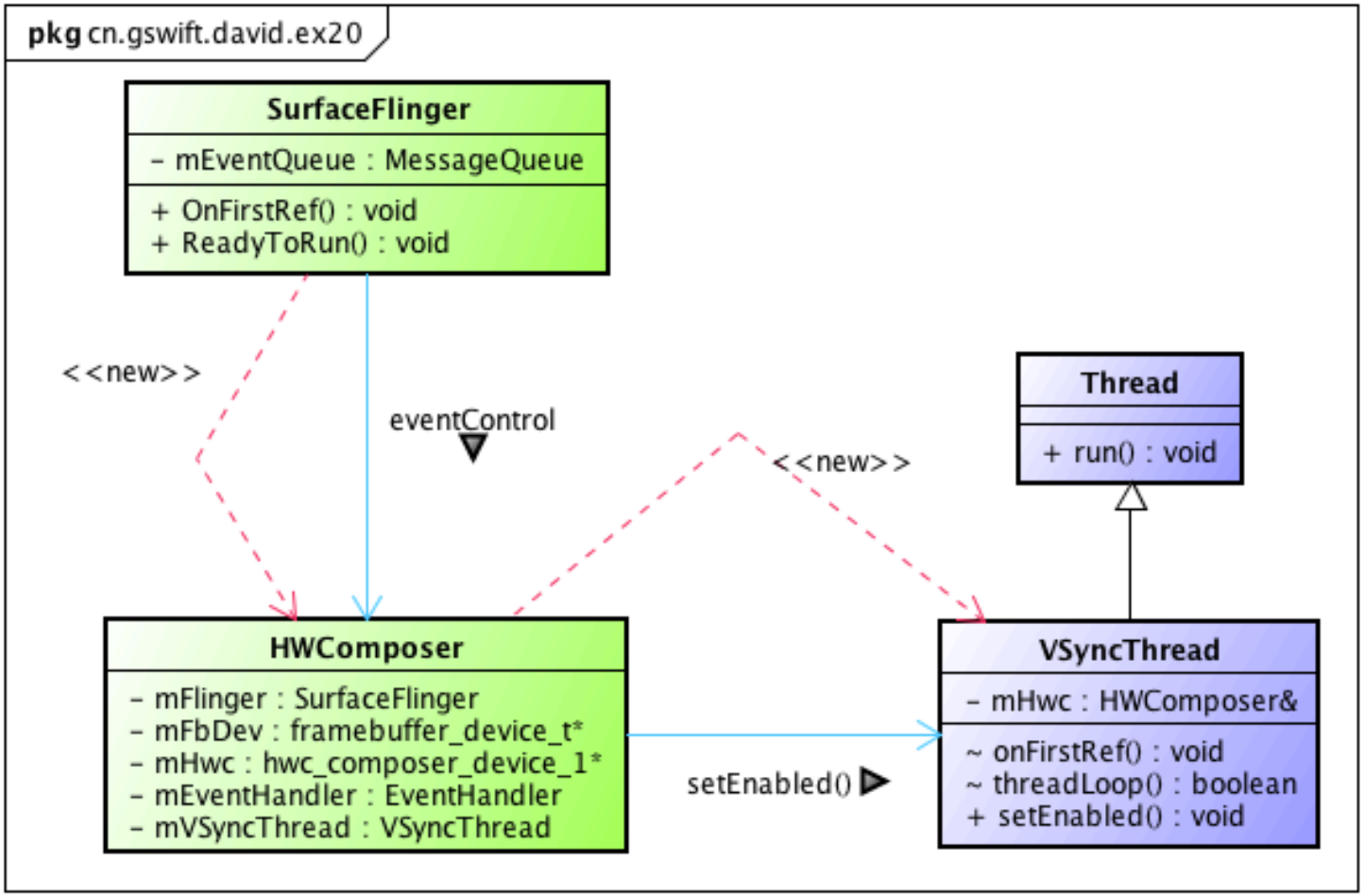
MessageQueue::setEventThread
Looper->addFd



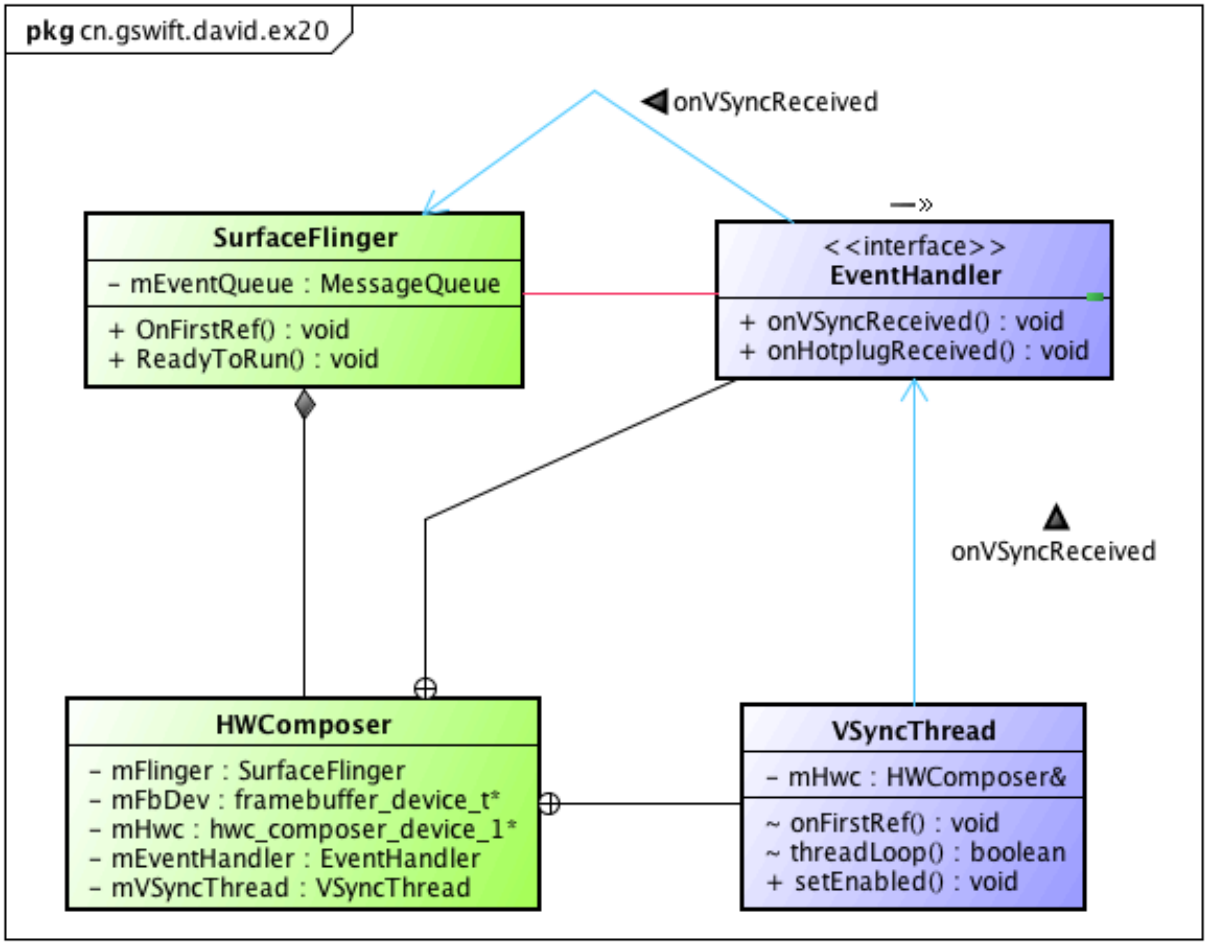
Looper::addFd()



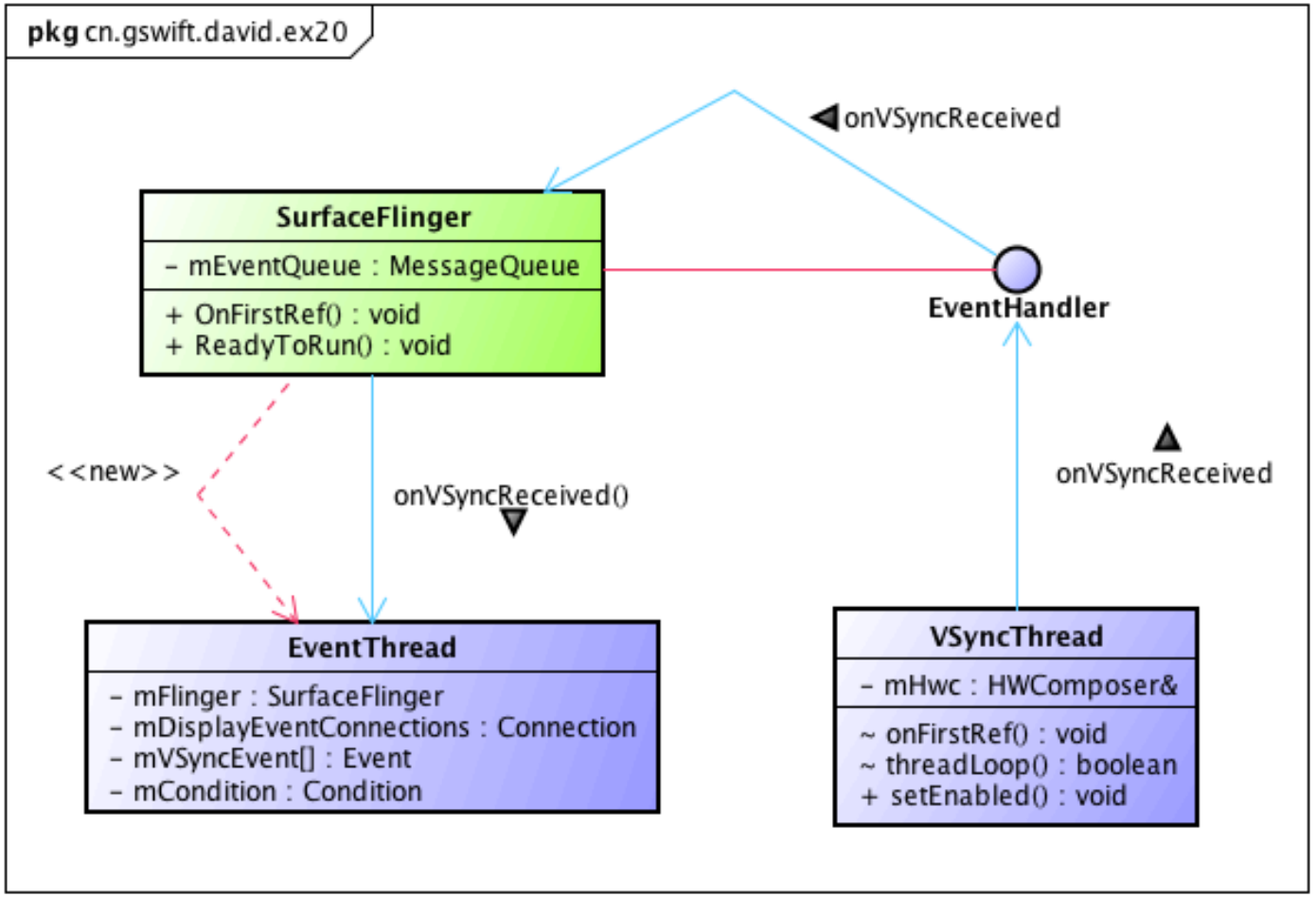
Vsync: VSyncThread



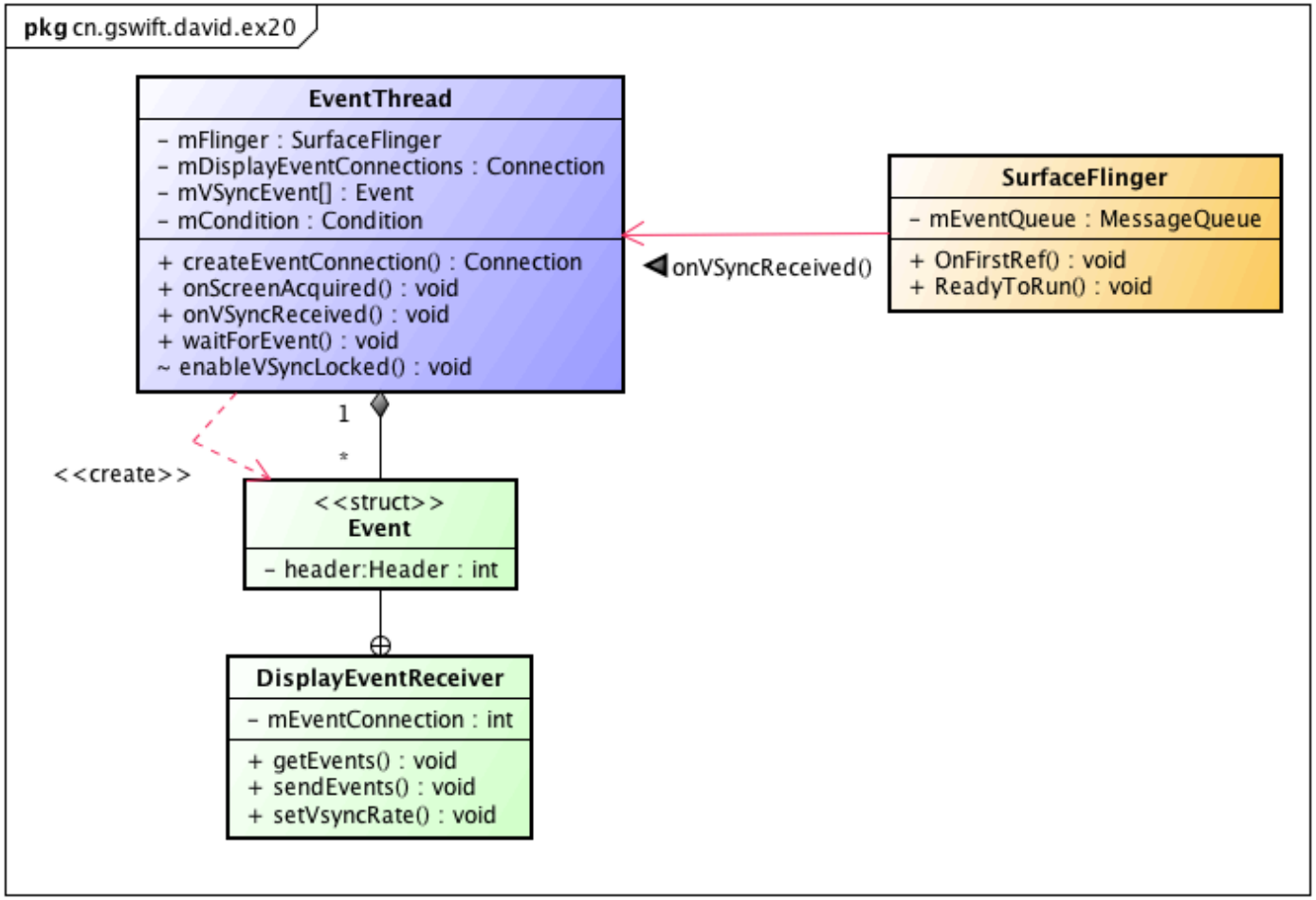
Vsync: onVsyncReceived()



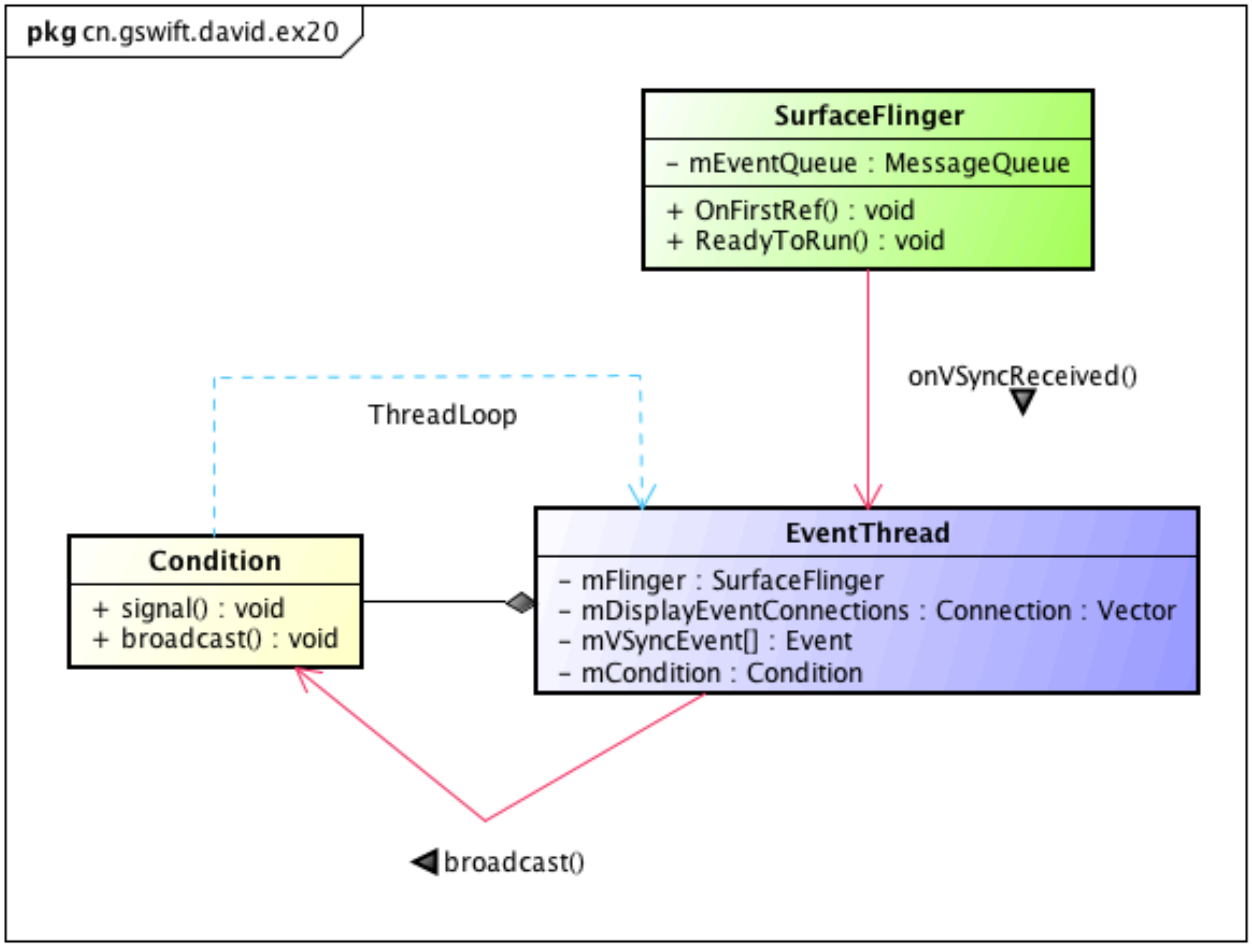
Vsync: onVsyncReceived()



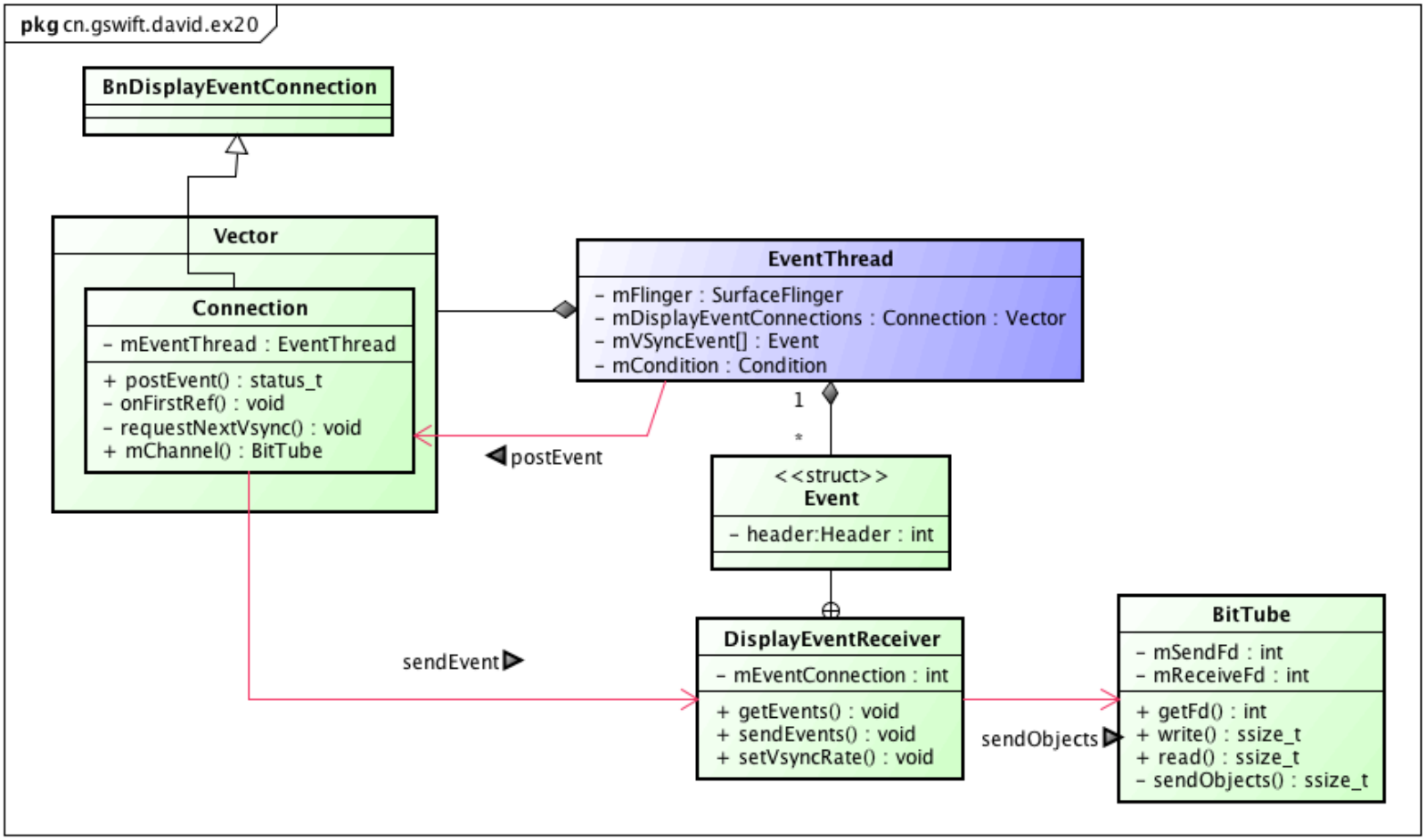
Vsync: onVsyncReceived()



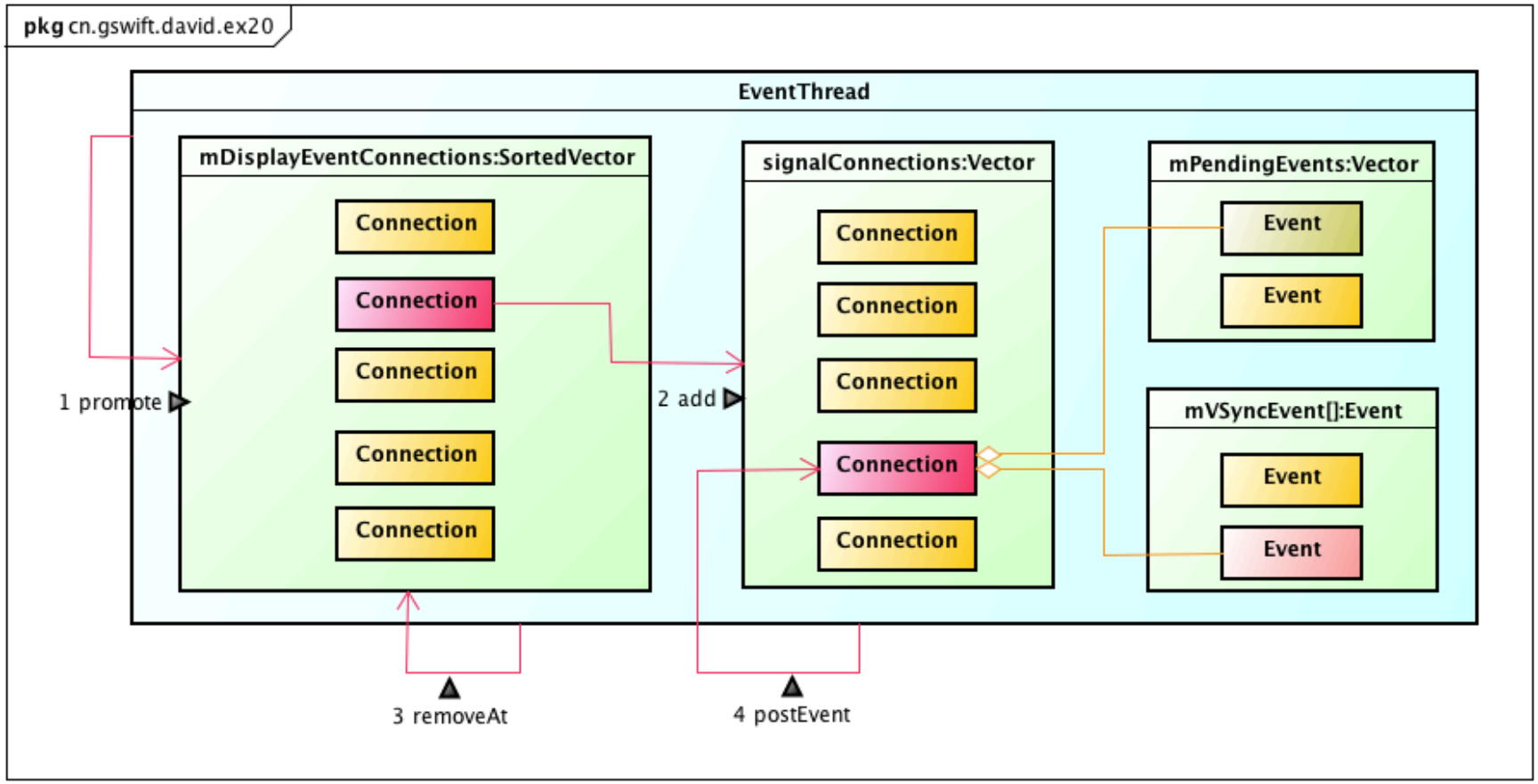
Vsync: onVsyncReceived()



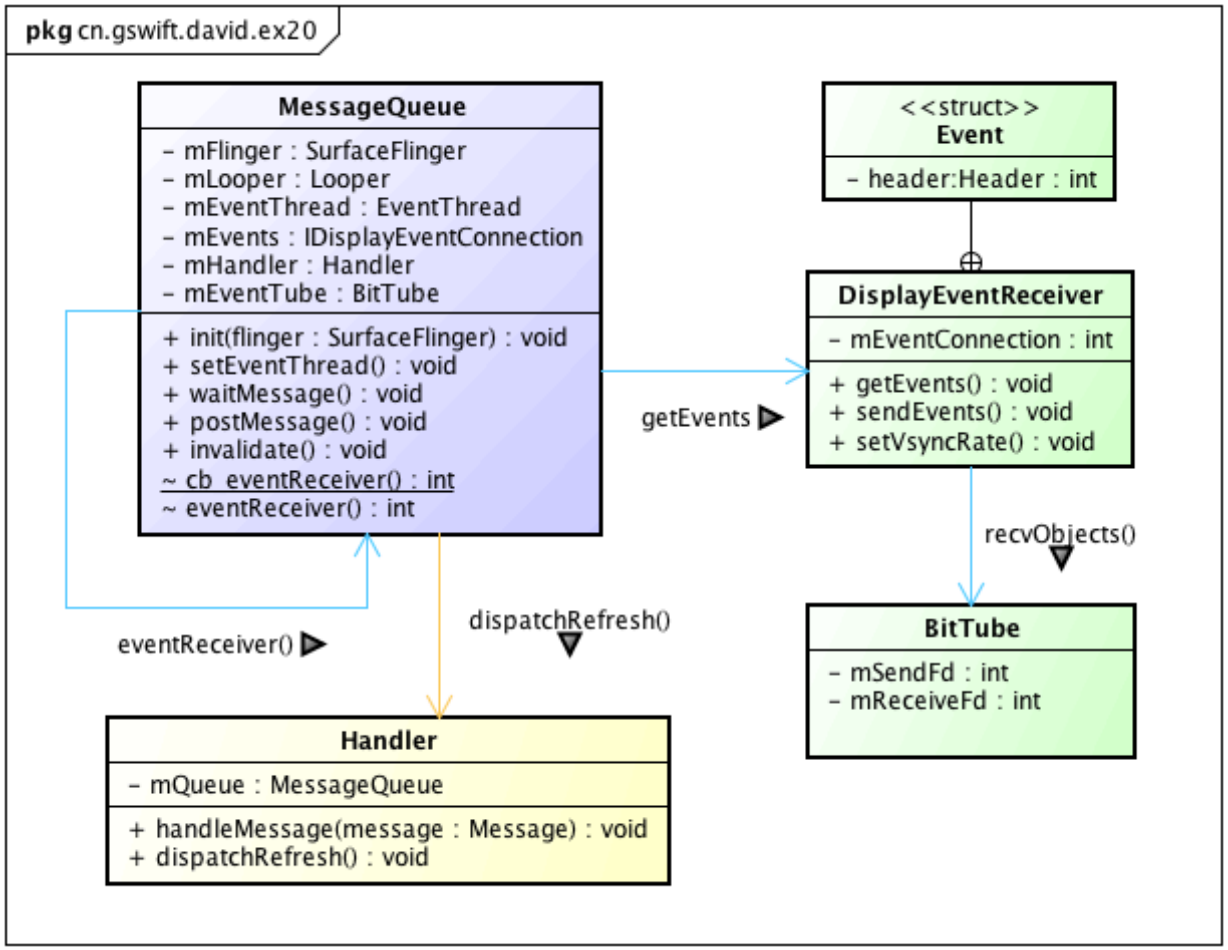
postEvent()


pkg cn.gswift.david.ex20
BnDisplayEventConnection
Vector
Connection
- mEventThread : EventThread
+ postEvent() : status_t
- onFirstRef() : void
- requestNextVsync() : void
+ mChannel() : BitTube
EventThread
- mFlinger : SurfaceFlinger
- mDisplayEventConnections : Connection : Vector
- mVSyncEvent[] : Event
- mCondition : Condition
1
*
postEvent
<<struct>>
Event
- header:Header : int
sendEvent
DisplayEventReceiver
- mEventConnection : int
+ getEvents() : void
+ sendEvents() : void
+ setVsyncRate() : void
sendObjects
BitTube
- mSendFd : int
- mReceiveFd : int
+ getFd() : int
+ write() : ssize_t
+ read() : ssize_t
- sendObjects() : ssize_t

postEvent()


pkg cn.gswift.david.ex20
EventThread
mDisplayEventConnections:SortedVector
Connection
Connection
Connection
Connection
Connection
signalConnections:Vector
Connection
Connection
Connection
Connection
Connection
mPendingEvents:Vector
Event
Event
mVSyncEvent[]:Event
Event
Event
1 promote
2 add
3 removeAt
4 postEvent

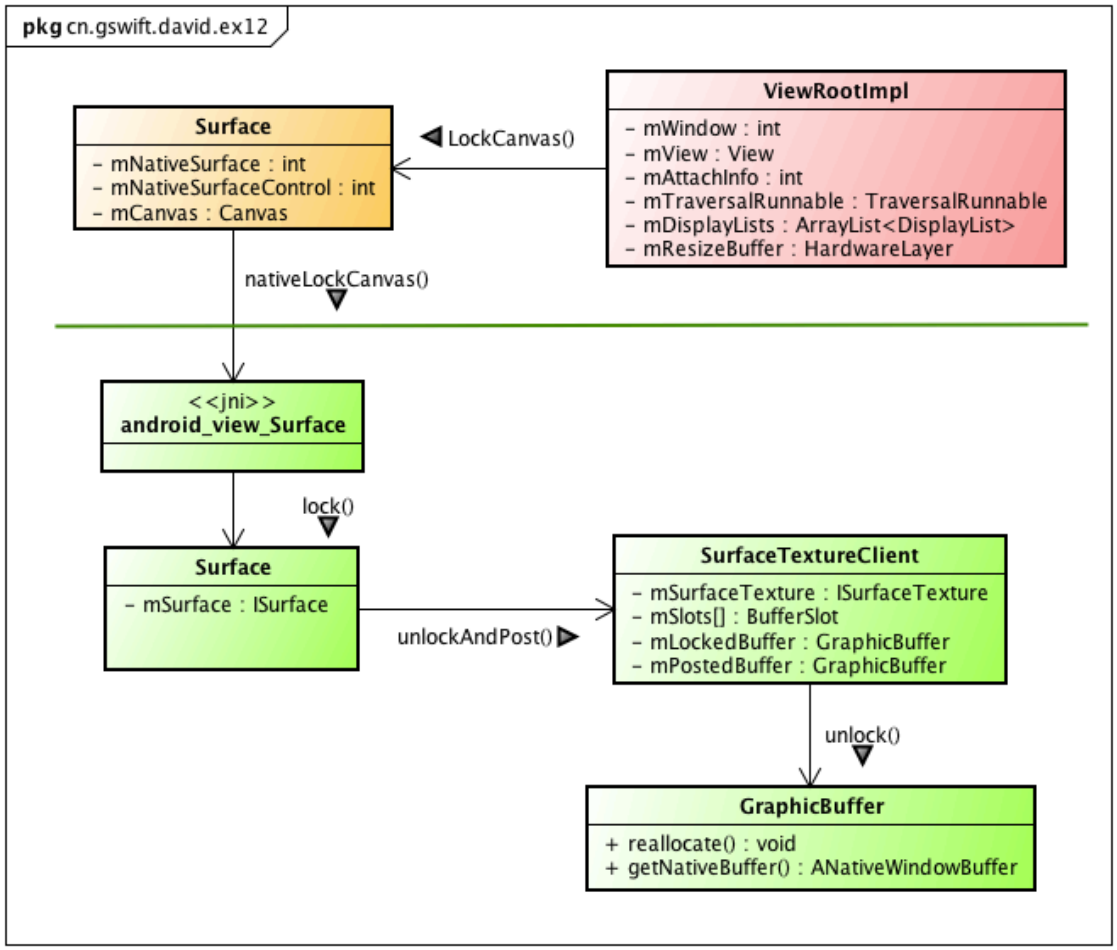
MessageQueue::setEventThread<3>
cb_eventReceiver



Handle Transaction

SurfaceFlinger

lockCanvas()


pkg cn.gswift.david.ex12
Surface
- mNativeSurface : int
- mNativeSurfaceControl : int
- mCanvas : Canvas
LockCanvas()
ViewRootImpl
- mWindow : int
- mView : View
- mAttachInfo : int
- mTraversalRunnable : TraversalRunnable
- mDisplayLists : ArrayList<DisplayList>
- mResizeBuffer : HardwareLayer
nativeLockCanvas()
<<jni>>
android_view_Surface
lock()
Surface
- mSurface : ISurface
unlockAndPost()
SurfaceTextureClient
- mSurfaceTexture : ISurfaceTexture
- mSlots[] : BufferSlot
- mLockedBuffer : GraphicBuffer
- mPostedBuffer : GraphicBuffer
unlock()
GraphicBuffer
+ reallocate() : void
+ getNativeBuffer() : ANativeWindowBuffer

Z-order

```
if ((mAttrs.type >= FIRST_SUB_WINDOW &&
        mAttrs.type <= LAST_SUB_WINDOW)) {
    // The multiplier here is to reserve space for multiple
    // windows in the same type layer.
    mBaseLayer = mPolicy.windowTypeToLayerLw(
            attachedWindow.mAttrs.type) * WindowManagerService.TYPE_LAYER_MULTIPLIER
            + WindowManagerService.TYPE_LAYER_OFFSET;
    mSubLayer = mPolicy.subWindowTypeToLayerLw(a.type);
    mAttachedWindow = attachedWindow;
    if (WindowManagerService.DEBUG_ADD_REMOVE) Slog.v(TAG, "Adding " + this + " to " + mAttachedWindow);
    mAttachedWindow.mChildWindows.add(this);
    mLayoutAttached = mAttrs.type !=
            WindowManager.LayoutParams.TYPE_APPLICATION_ATTACHED_DIALOG;
    mIsImWindow = attachedWindow.mAttrs.type == TYPE_INPUT_METHOD
            || attachedWindow.mAttrs.type == TYPE_INPUT_METHOD_DIALOG;
    mIsWallpaper = attachedWindow.mAttrs.type == TYPE_WALLPAPER;
    mIsFloatingLayer = mIsImWindow || mIsWallpaper;
} else {
    // The multiplier here is to reserve space for multiple
    // windows in the same type layer.
    mBaseLayer = mPolicy.windowTypeToLayerLw(a.type)
            * WindowManagerService.TYPE_LAYER_MULTIPLIER
            + WindowManagerService.TYPE_LAYER_OFFSET;
    mSubLayer = 0;
    mAttachedWindow = null;
    mLayoutAttached = false;
    mIsImWindow = mAttrs.type == TYPE_INPUT_METHOD
            || mAttrs.type == TYPE_INPUT_METHOD_DIALOG;
    mIsWallpaper = mAttrs.type == TYPE_WALLPAPER;
    mIsFloatingLayer = mIsImWindow || mIsWallpaper;
}
```

Each window is also a layer. The layers are sorted by Z-order. The Z-order is just the layer type as specified in PhoneWindowManager.java. When adding a layer with a Z-order that is already used by some other layer in SurfaceFlinger's list of layers it is put on top of the layers with the same Z-order.

wm/WindowStateAnimator.java

```
Surface.openTransaction();
try {
    try {
        mSurfaceX = mWin.mFrame.left + mWin.mXOffset;
        mSurfaceY = mWin.mFrame.top + mWin.mYOffset;
        mSurface.setPosition(mSurfaceX, mSurfaceY);
        mSurfaceLayer = mAnimLayer;
        mSurface.setLayerStack(mLayerStack);
        mSurface.setLayer(mAnimLayer);
        mSurface.setAlpha(0);
        mSurfaceShown = false;
    } catch (RuntimeException e) {
        Slog.w(TAG, "Error creating surface in " + w, e);
        mService.reclaimSomeSurfaceMemoryLocked(this, "create-init", true);
    }
    mLastHidden = true;
} finally {
    Surface.closeTransaction();
    if (SHOW_LIGHT_TRANSACTIONS) Slog.i(TAG,
            "<<< CLOSE TRANSACTION createSurfaceLocked");
}
```

jni/android_view_Surface.cpp

```
static void nativeSetLayer(JNIEnv* env, jobject surfaceObj, jint zorder) {
    sp<SurfaceControl> surface(getSurfaceControl(env, surfaceObj));
    if (surface == NULL) return;

    status_t err = surface->setLayer(zorder);
    if (err < 0 && err != NO_INIT) {
        doThrowIAE(env);
    }
}
```

```
public WindowStateAnimator(final WindowState win) {
    final WindowManagerService service = win.mService;

    mService = service;
    mAnimator = service.mAnimator;
    mPolicy = service.mPolicy;
    mContext = service.mContext;
    final DisplayInfo displayInfo = win.mDisplayContent.getDisplayInfo();
    mAnimDw = displayInfo.appWidth;
    mAnimDh = displayInfo.appHeight;

    mWin = win;
    mAttachedWinAnimator = win.mAttachedWindow == null
            ? null : win.mAttachedWindow.mWinAnimator;
    mAppAnimator = win.mAppToken == null ? null : win.mAppToken.mAppAnimator;
    mSession = win.mSession;
    mAttrFlags = win.mAttrs.flags;
    mAttrType = win.mAttrs.type;
    mIsWallpaper = win.mIsWallpaper;
    mLayerStack = win.mDisplayContent.getDisplay().getLayerStack();
}
```

```
Surface.openTransaction();
try {
    try {
        mSurfaceX = mWin.mFrame.left + mWin.mXOffset;
        mSurfaceY = mWin.mFrame.top + mWin.mYOffset;
        mSurface.setPosition(mSurfaceX, mSurfaceY);
        mSurfaceLayer = mAnimLayer;
        mSurface.setLayerStack(mLayerStack);
        mSurface.setLayer(mAnimLayer);
        mSurface.setAlpha(0);
        mSurfaceShown = false;
    } catch (RuntimeException e) {
        Slog.w(TAG, "Error creating surface in " + w, e);
        mService.reclaimSomeSurfaceMemoryLocked(this, "create-init", true);
    }
    mLastHidden = true;
} finally {
    Surface.closeTransaction();
    if (SHOW_LIGHT_TRANSACTIONS) Slog.i(TAG,
            "<<< CLOSE TRANSACTION createSurfaceLocked");
}
```

Source:/android/4.2/frameworks/base/services/java/com/android/server/wm/WindowStateAnimator.java

```
public WindowStateAnimator(final WindowState win) {
    final WindowManagerService service = win.mService;

    mService = service;
    mAnimator = service.mAnimator;
    mPolicy = service.mPolicy;
    mContext = service.mContext;
    final DisplayInfo displayInfo = win.mDisplayContent.getDisplayInfo();
    mAnimDw = displayInfo.appWidth;
    mAnimDh = displayInfo.appHeight;

    mWin = win;
    mAttachedWinAnimator = win.mAttachedWindow == null
            ? null : win.mAttachedWindow.mWinAnimator;
    mAppAnimator = win.mAppToken == null ? null : win.mAppToken.mAppAnimator;
    mSession = win.mSession;
    mAttrFlags = win.mAttrs.flags;
    mAttrType = win.mAttrs.type;
    mIsWallpaper = win.mIsWallpaper;
    mLayerStack = win.mDisplayContent.getDisplay().getLayerStack();
}
```

```
Surface.openTransaction();
try {
    try {
        mSurfaceX = mWin.mFrame.left + mWin.mXOffset;
        mSurfaceY = mWin.mFrame.top + mWin.mYOffset;
        mSurface.setPosition(mSurfaceX, mSurfaceY);
        mSurfaceLayer = mAnimLayer;
        mSurface.setLayerStack(mLayerStack);
        mSurface.setLayer(mAnimLayer);
        mSurface.setAlpha(0);
        mSurfaceShown = false;
    } catch (RuntimeException e) {
        Slog.w(TAG, "Error creating surface in " + w, e);
        mService.reclaimSomeSurfaceMemoryLocked(this, "create-init", true);
    }
    mLastHidden = true;
} finally {
    Surface.closeTransaction();
    if (SHOW_LIGHT_TRANSACTIONS) Slog.i(TAG,
            "<<< CLOSE TRANSACTION createSurfaceLocked");
}
```

Source:/android/4.2/frameworks/base/services/java/com/android/server/wm/WindowStateAnimator.java

wm/WindowStateAnimator.java

```
Surface.openTransaction();
try {
    try {
        mSurfaceX = mWin.mFrame.left + mWin.mXOffset;
        mSurfaceY = mWin.mFrame.top + mWin.mYOffset;
        mSurface.setPosition(mSurfaceX, mSurfaceY);
        mSurfaceLayer = mAnimLayer;
        mSurface.setLayerStack(mLayerStack);
        mSurface.setLayer(mAnimLayer);
        mSurface.setAlpha(0);
        mSurfaceShown = false;
    } catch (RuntimeException e) {
        Slog.w(TAG, "Error creating surface in " + w, e);
        mService.reclaimSomeSurfaceMemoryLocked(this, "create-init", true);
    }
    mLastHidden = true;
} finally {
    Surface.closeTransaction();
    if (SHOW_LIGHT_TRANSACTIONS) Slog.i(TAG,
            "<<< CLOSE TRANSACTION createSurfaceLocked");
}
```

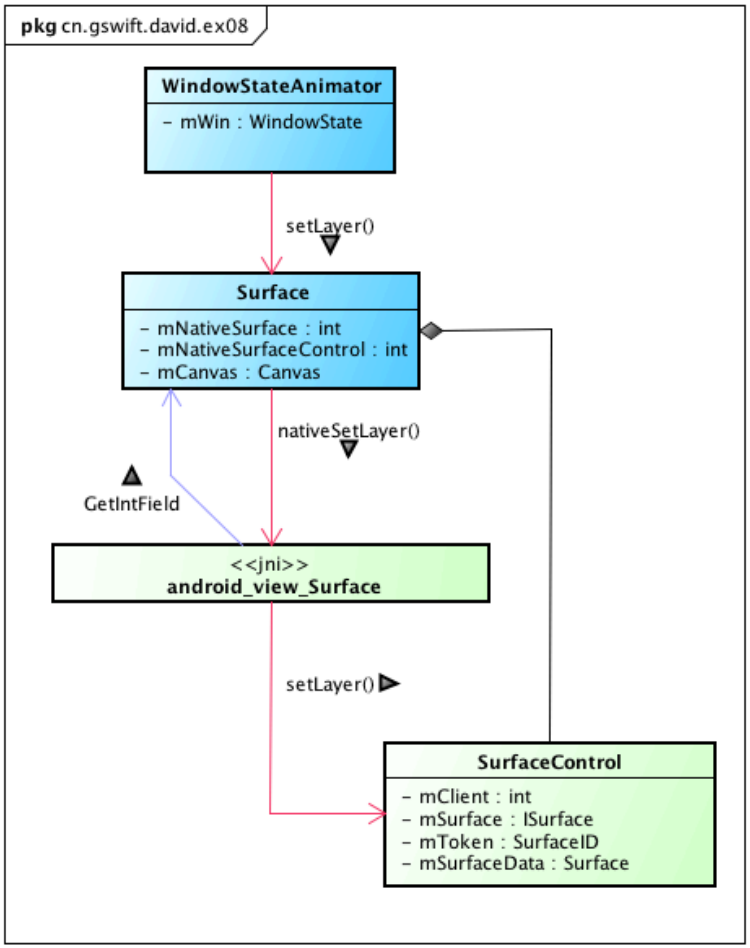
jni/android_view_Surface.cpp

```
static void nativeSetLayerStack(JNIEnv* env, jobject surfaceObj, jint layerStack) {
    sp<SurfaceControl> surface(getSurfaceControl(env, surfaceObj));
    if (surface == NULL) return;

    status_t err = surface->setLayerStack(layerStack);
    if (err < 0 && err != NO_INIT) {
        doThrowIAE(env);
    }
}
```

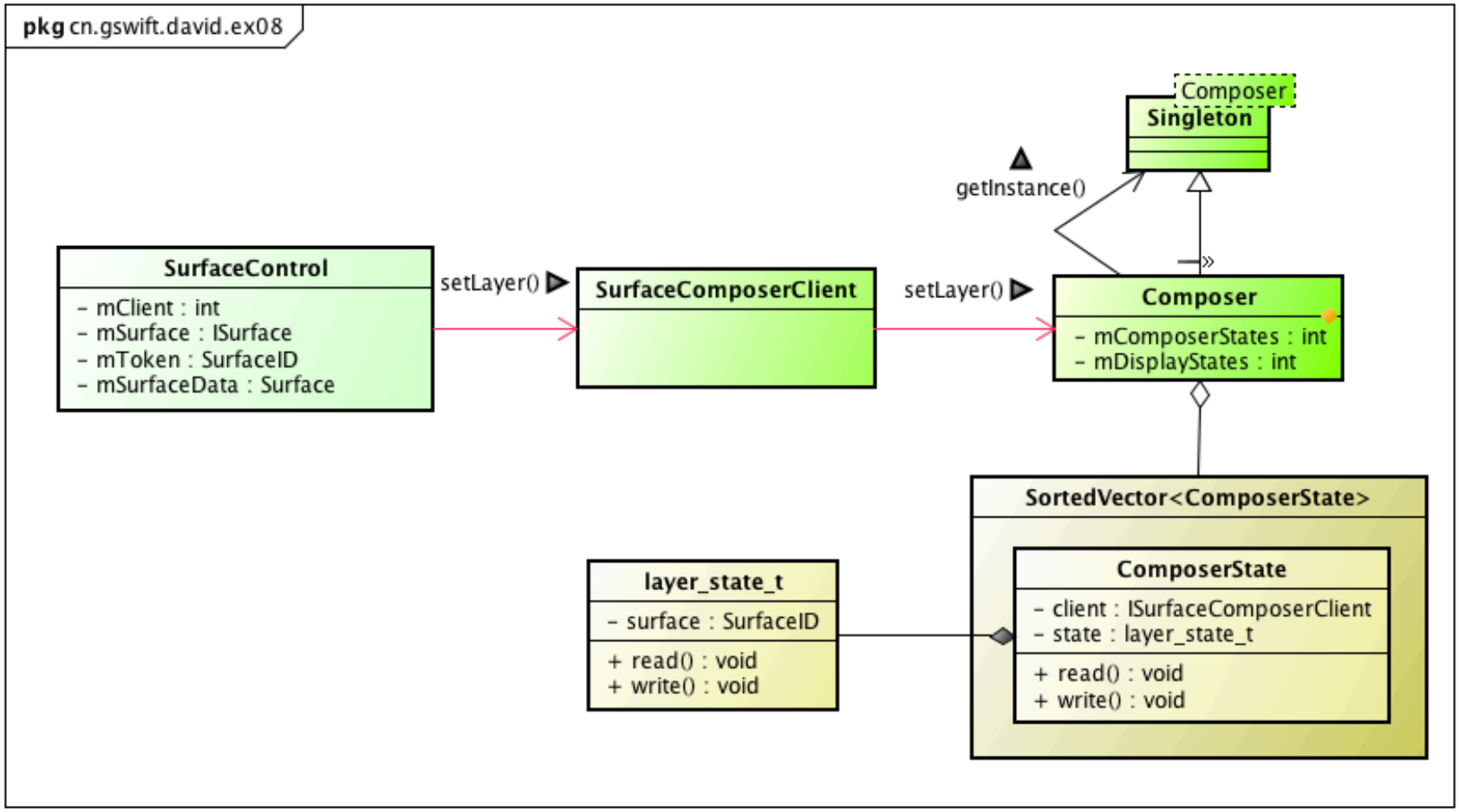
```
status_t Composer::setLayer(const sp<SurfaceComposerClient>& client,
        SurfaceID id, int32_t z) {
    Mutex::Autolock _l(mLock);
    layer_state_t* s = getLayerStateLocked(client, id);
    if (!s)
        return BAD_INDEX;
    s->what |= layer_state_t::eLayerChanged;
    s->z = z;
    return NO_ERROR;
}

status_t Composer::setLayerStack(const sp<SurfaceComposerClient>& client,
        SurfaceID id, uint32_t layerStack) {
    Mutex::Autolock _l(mLock);
    layer_state_t* s = getLayerStateLocked(client, id);
    if (!s)
        return BAD_INDEX;
    s->what |= layer_state_t::eLayerStackChanged;
    s->layerStack = layerStack;
    return NO_ERROR;
}
```

```
struct layer_state_t {

    enum {
        eLayerHidden        = 0x01,
    };

    enum {
        ePositionChanged            = 0x00000001,
        eLayerChanged               = 0x00000002,
        eSizeChanged                = 0x00000004,
        eAlphaChanged               = 0x00000008,
        eMatrixChanged              = 0x00000010,
        eTransparentRegionChanged   = 0x00000020,
        eVisibilityChanged          = 0x00000040,
        eLayerStackChanged          = 0x00000080,
        eCropChanged                = 0x00000100,
    };

    layer_state_t()
        :   surface(0), what(0),
            x(0), y(0), z(0), w(0), h(0), layerStack(0),
            alpha(0), flags(0), mask(0),
            reserved(0)
    {
        matrix.dsdx = matrix.dtdy = 1.0f;
        matrix.dsdy = matrix.dtdx = 0.0f;
        crop.makeInvalid();
    }

    status_t    write(Parcel& output) const;
    status_t    read(const Parcel& input);

            struct matrix22_t {
                float   dsdx;
                float   dtdx;
                float   dsdy;
                float   dtdy;
            };
            SurfaceID       surface;
            uint32_t        what;
            float           x;
            float           y;
            uint32_t        z;
            uint32_t        w;
            uint32_t        h;
            uint32_t        layerStack;
            float           alpha;
            uint8_t         flags;
            uint8_t         mask;
            uint8_t         reserved;
            matrix22_t      matrix;
            Rect            crop;
            // non POD must be last. see write/read
            Region          transparentRegion;
};
```

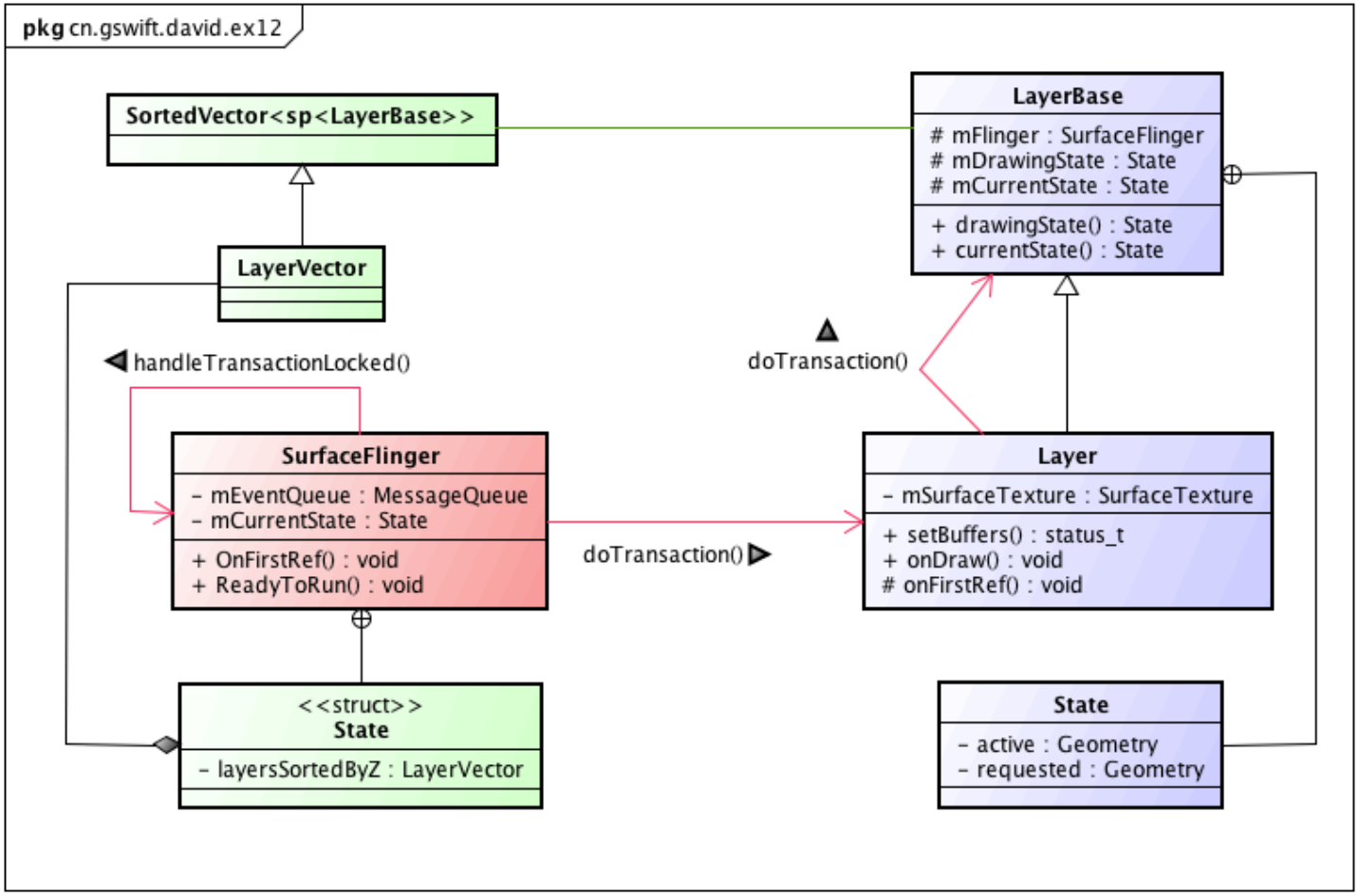
Transaction: setLayer(1)



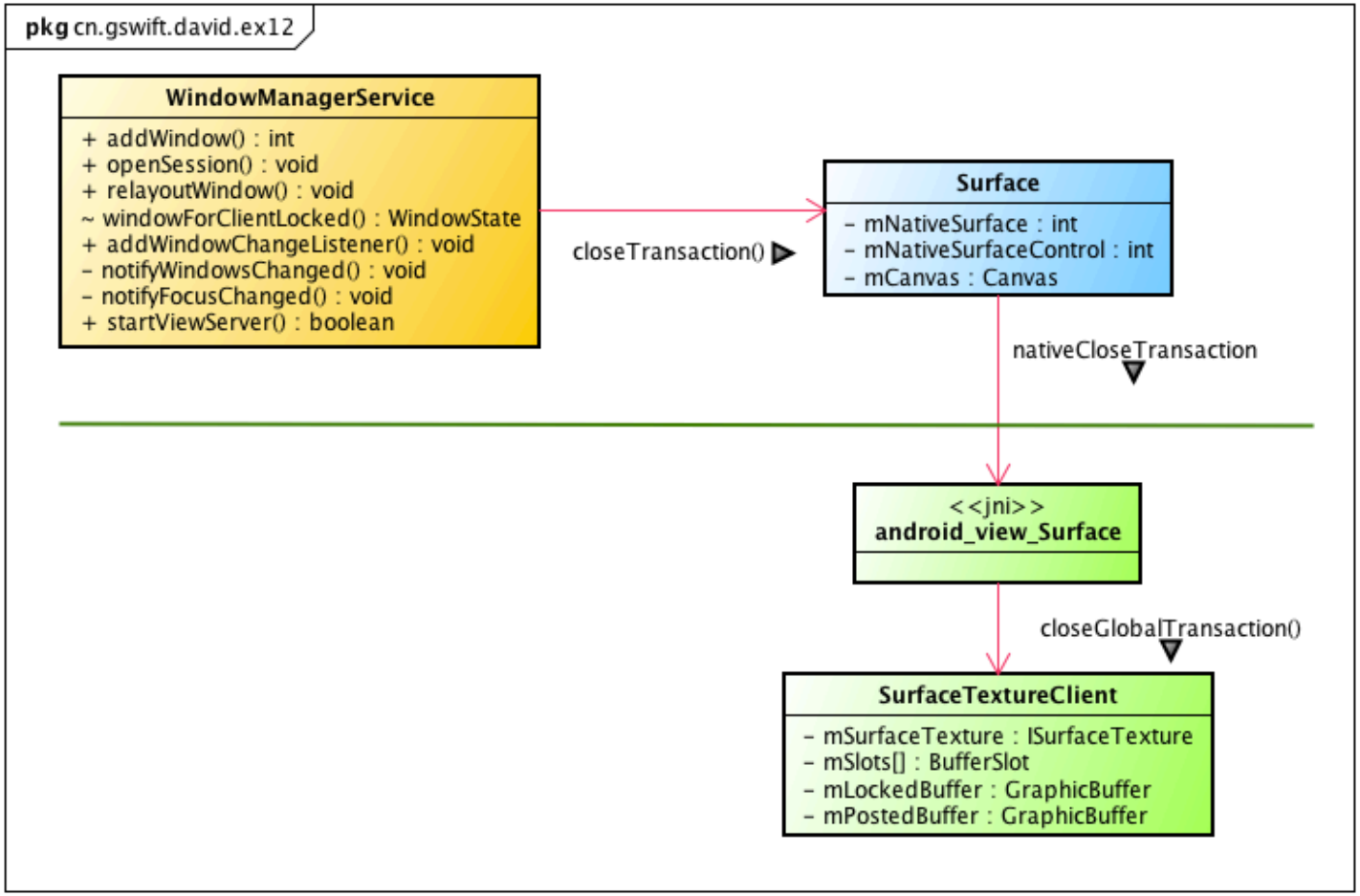
Transaction: setLayer(2)



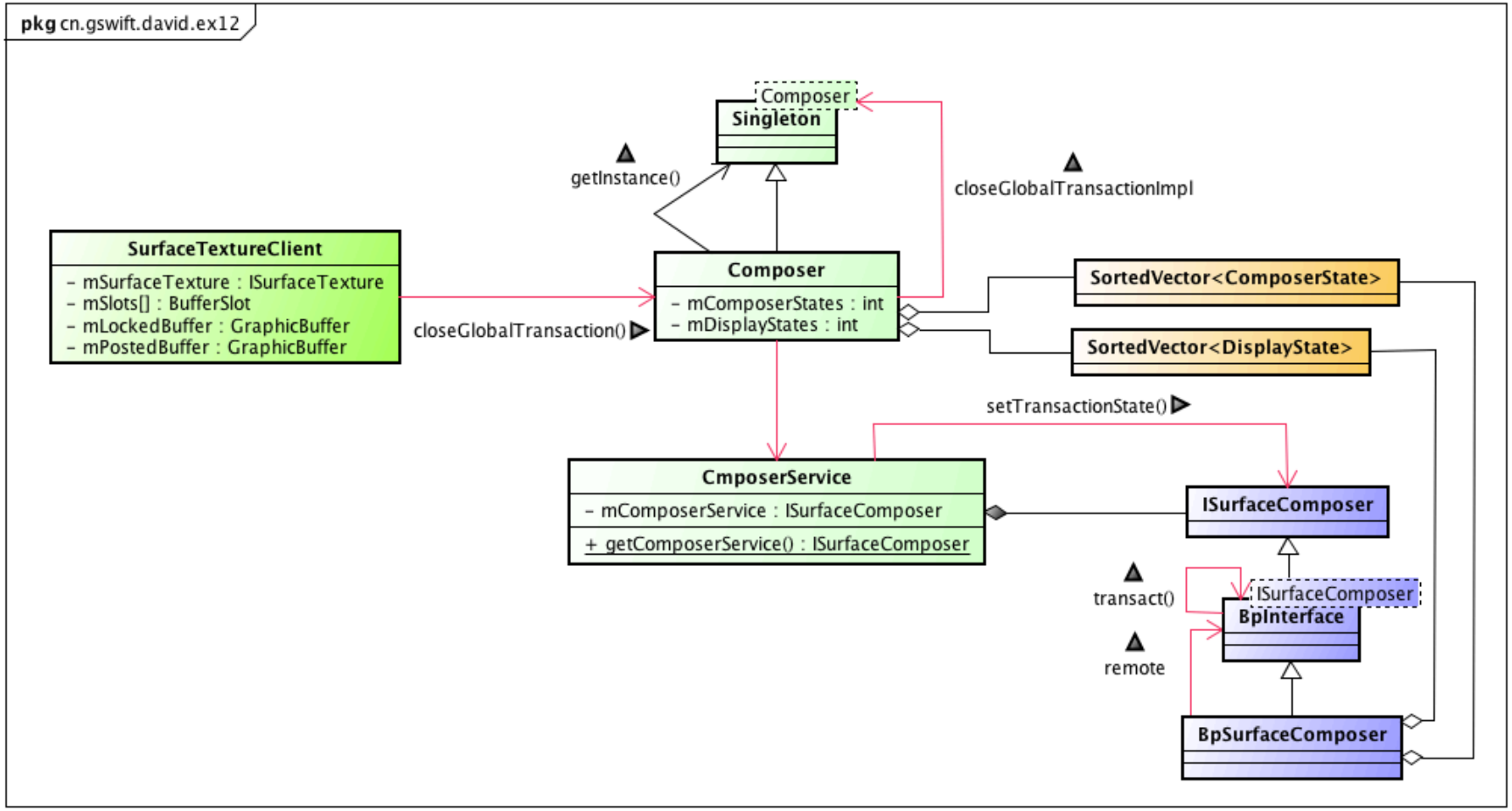
Handle Transaction for Layer



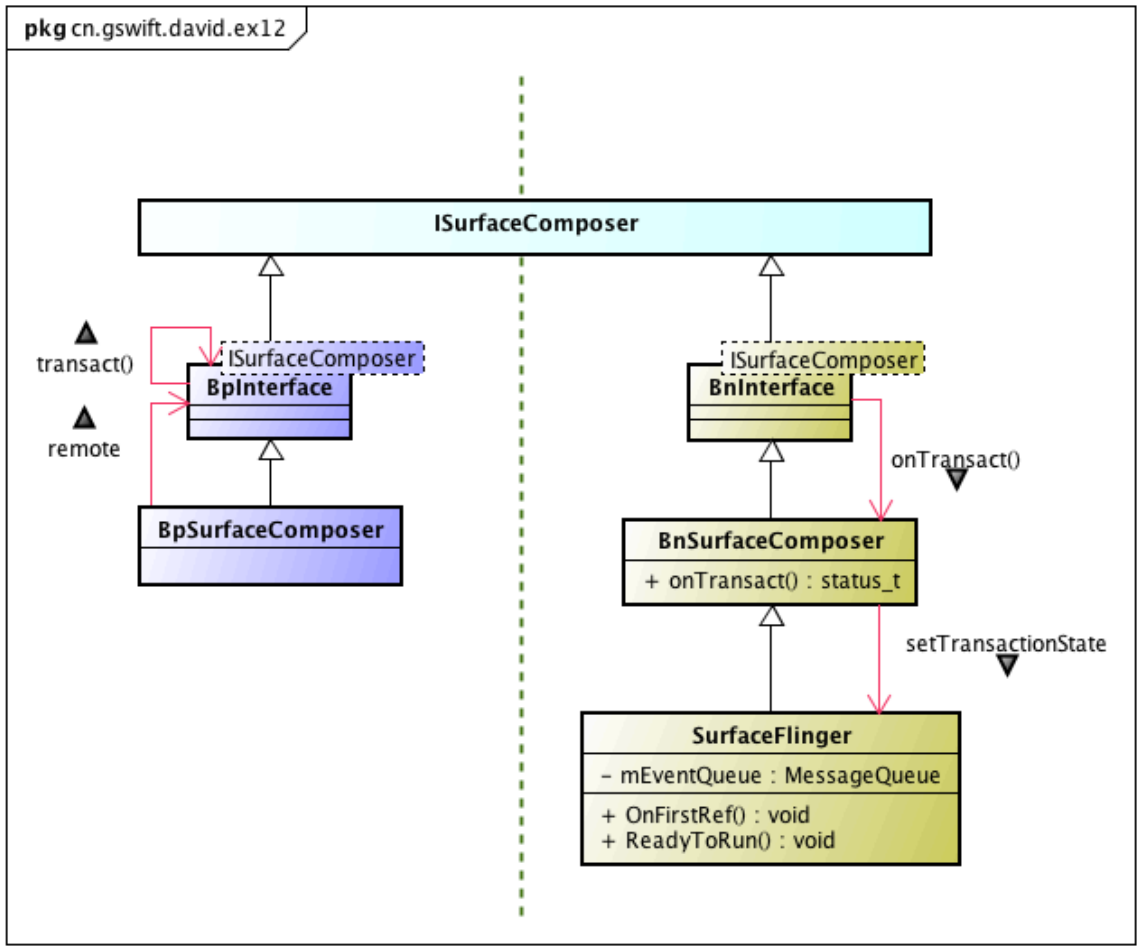
Close the Surface's Transaction(1)



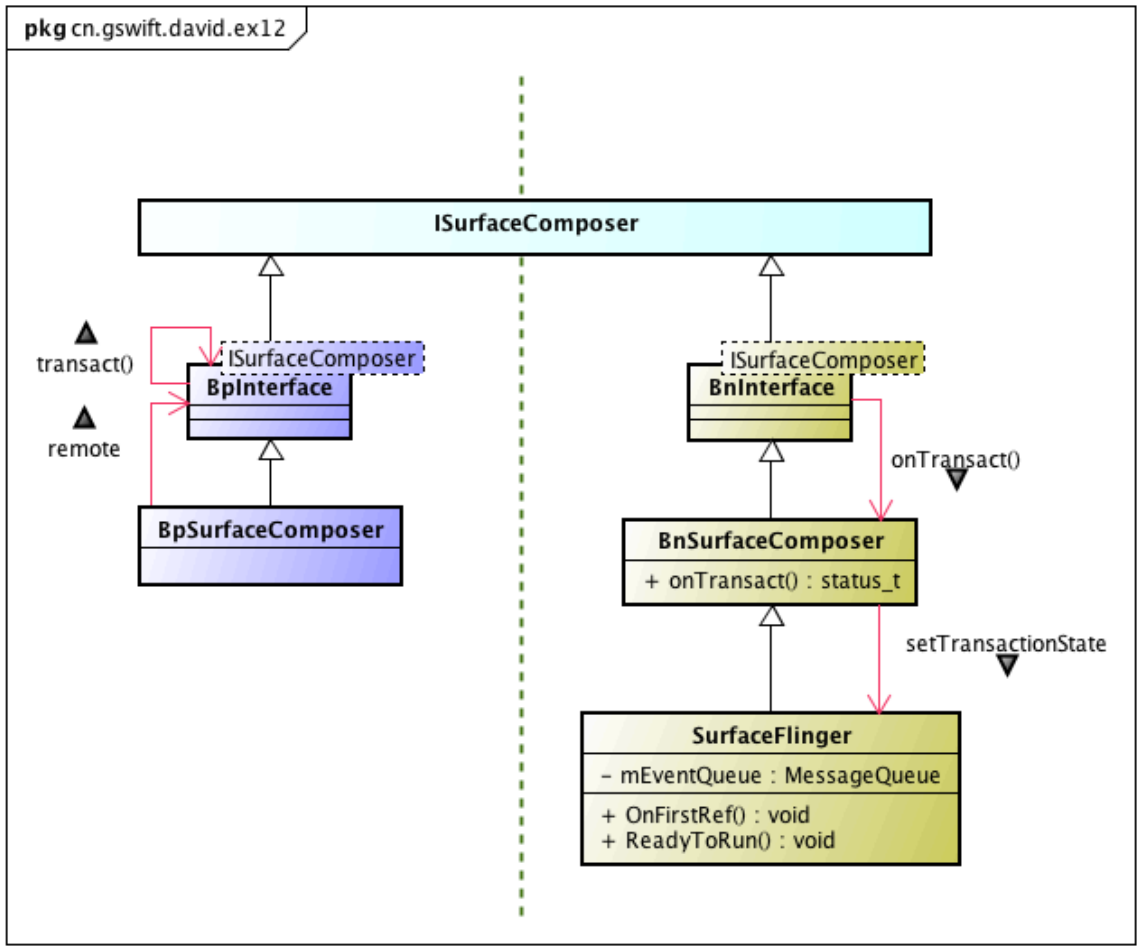
Close the Surface's Transaction(2)



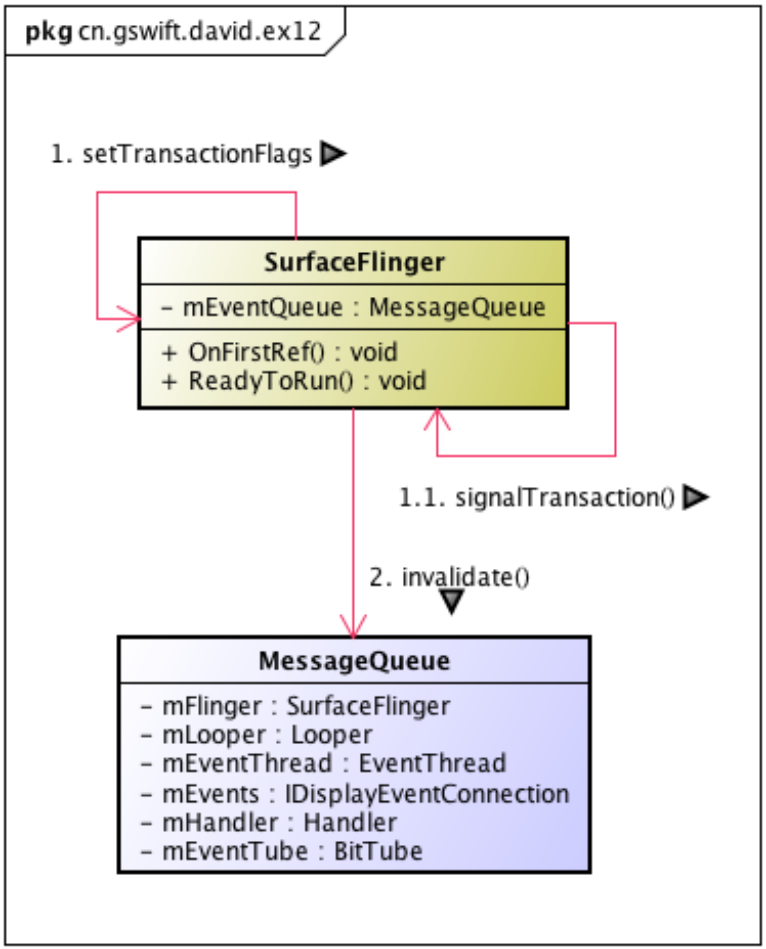
Close the Surface's Transaction(3)



Close the Surface's Transaction(4)



Close the Surface's Transaction(5)



HandleTransaction(2)

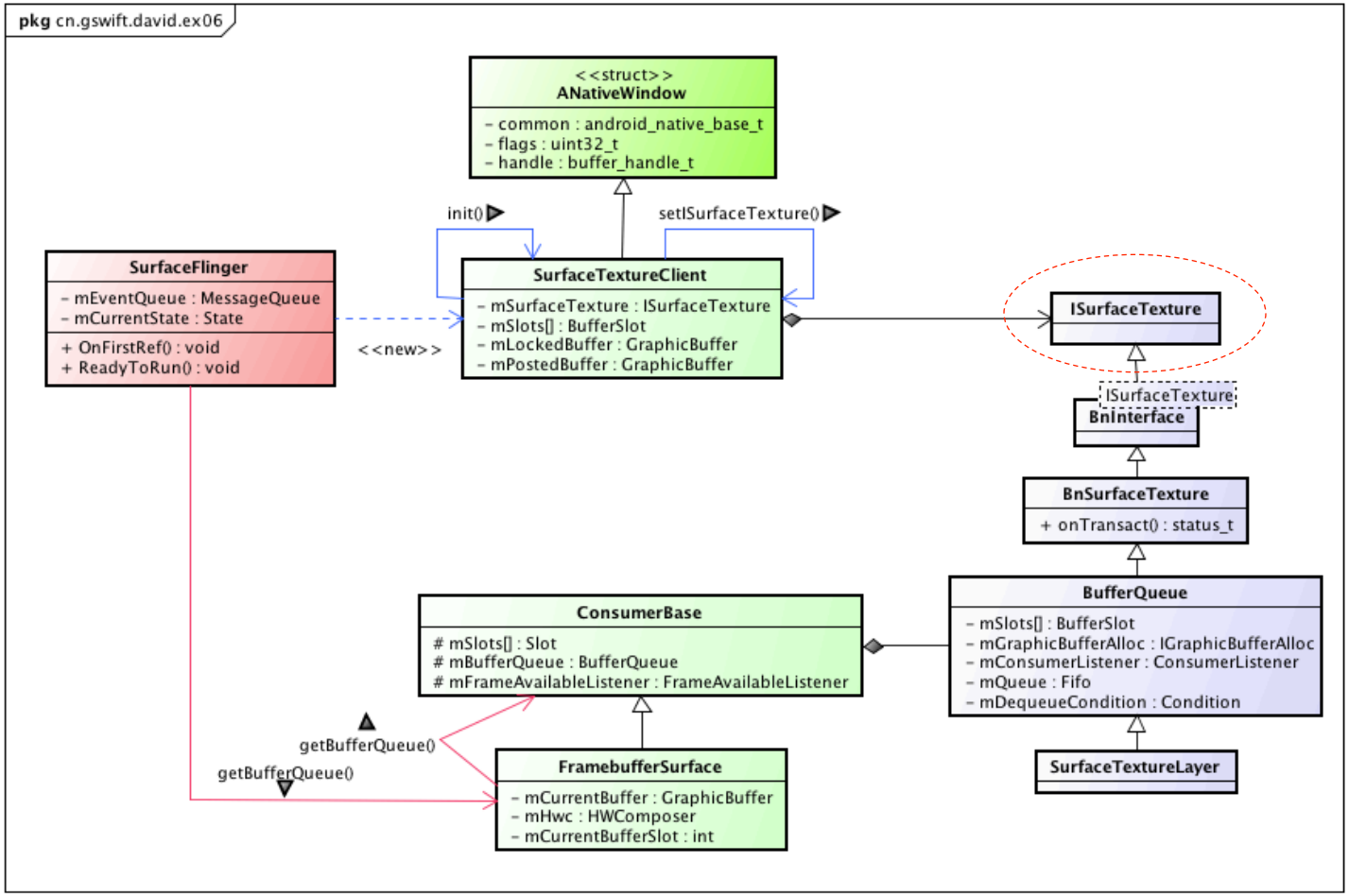
```
// find displays that were added
// (ie: in current state but not in drawing state)
for (size_t i=0 ; i<cc ; i++) {
    if (draw.indexOfKey(curr.keyAt(i)) < 0) {
        const DisplayDeviceState& state(curr[i]);
        bool isSecure = false;

        sp<FramebufferSurface> fbs;
        sp<SurfaceTextureClient> stc;
        if (!state.isVirtualDisplay()) {

            ALOGE_IF(state.surface!=NULL,
                    "adding a supported display, but rendering "
                    "surface is provided (%p), ignoring it",
                    state.surface.get());

            // All non-virtual displays are currently considered
            // secure.
            isSecure = true;

            // for supported (by hwc) displays we provide our
            // own rendering surface
            fbs = new FramebufferSurface(*mHwc, state.type);
            stc = new SurfaceTextureClient(
                    static_cast< sp<ISurfaceTexture> >(
                            fbs->getBufferQueue()));
        } else {
            if (state.surface != NULL) {
                stc = new SurfaceTextureClient(state.surface);
            }
            isSecure = state.isSecure;
        }

        const wp<IBinder>& display(curr.keyAt(i));
        if (stc != NULL) {
            sp<DisplayDevice> hw = new DisplayDevice(this,
                    state.type, isSecure, display, stc, fbs,
                    mEGLConfig);
            hw->setLayerStack(state.layerStack);
            hw->setProjection(state.orientation,
                    state.viewport, state.frame);
            hw->setDisplayName(state.displayName);
            mDisplays.add(display, hw);
            mEventThread->onHotplugReceived(state.type, true);
        }
    }
}
```

Create a FramebufferSurface instance

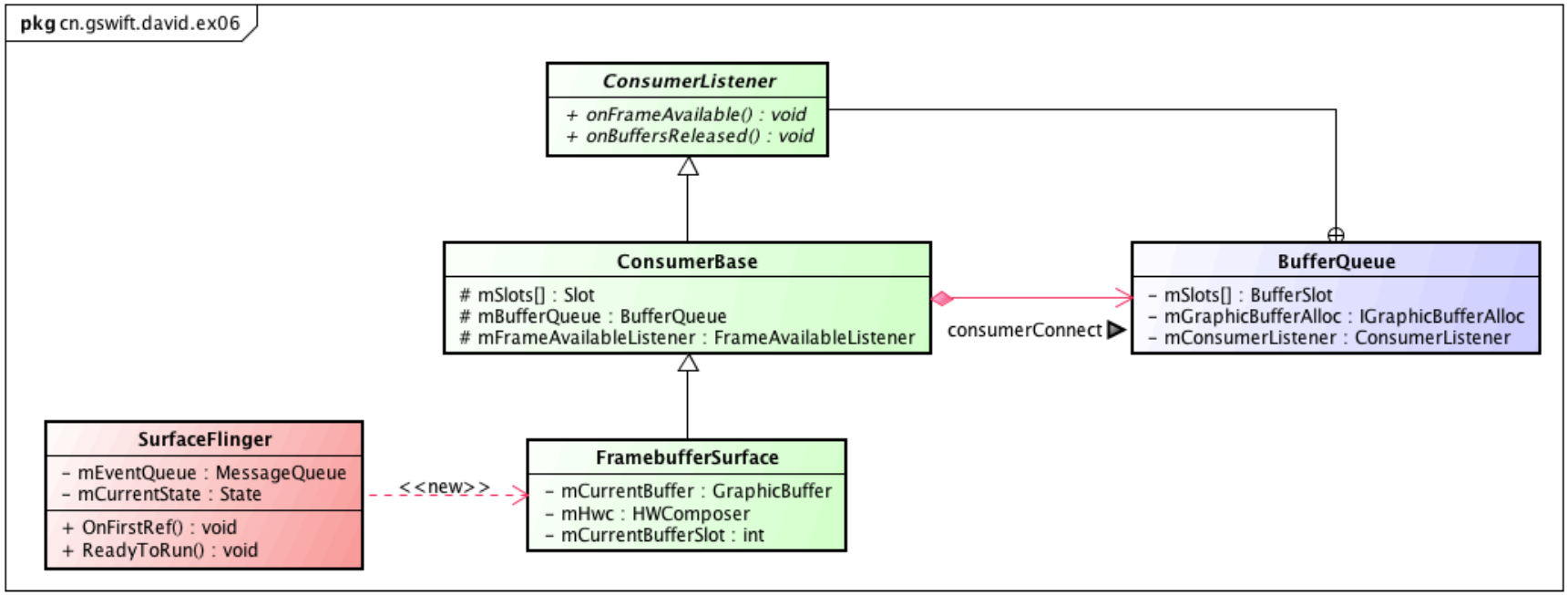
Create a SurfaceTextureClient instance

Create a DisplayDevice instance

Create a SurfaceTextureClient instance



Create a FrameBufferSurface instance



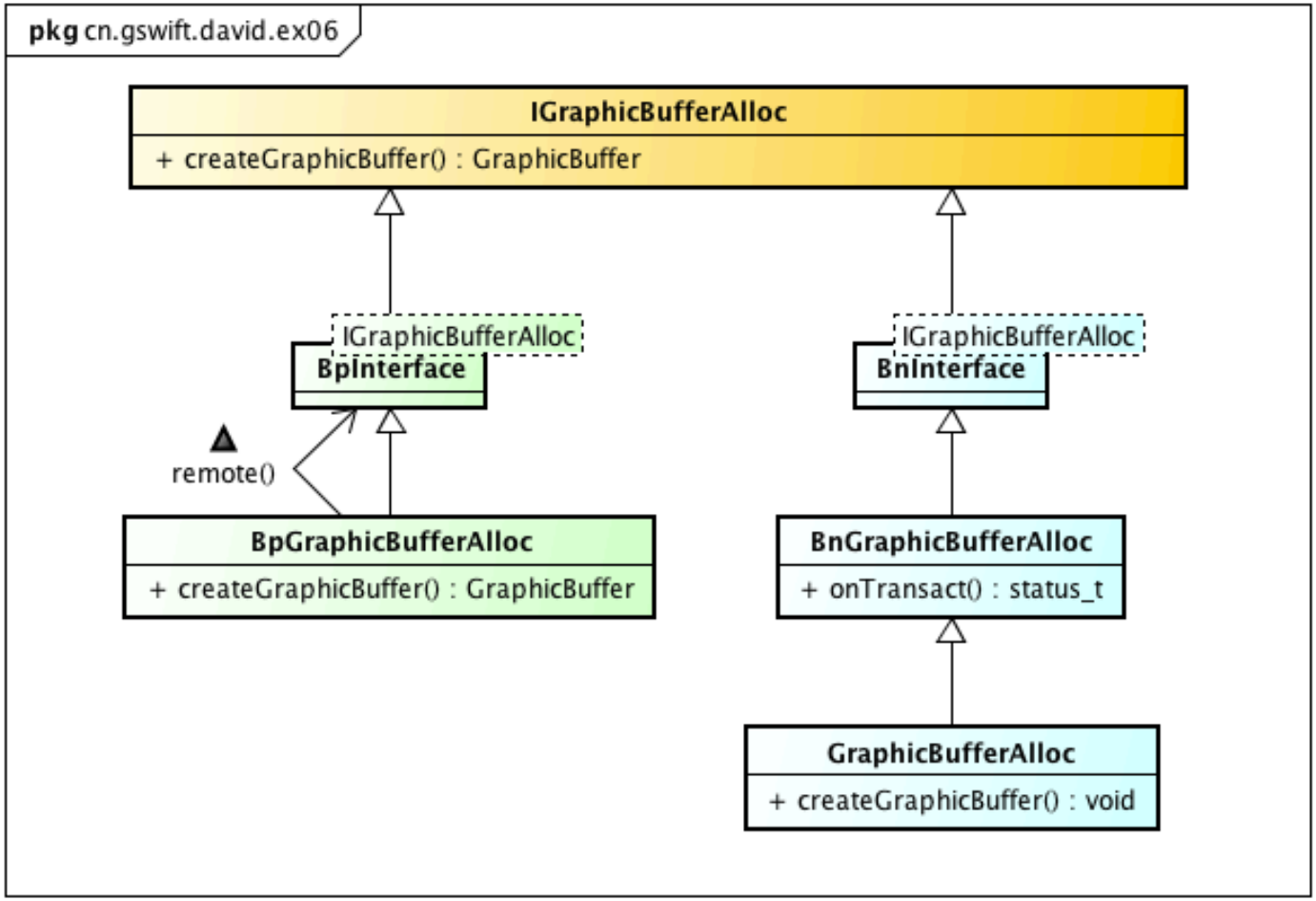
```
FramebufferSurface::FramebufferSurface(HWComposer& hwc, int disp) :
    ConsumerBase(new BufferQueue(true, new GraphicBufferAlloc())),
    mDisplayType(disp),
    mCurrentBufferSlot(-1),
    mCurrentBuffer(0),
    mHwc(hwc)
{
    mName = "FramebufferSurface";
    mBufferQueue->setConsumerName(mName);
    mBufferQueue->setConsumerUsageBits(GRALLOC_USAGE_HW_FB |
                                       GRALLOC_USAGE_HW_RENDER |
                                       GRALLOC_USAGE_HW_COMPOSER);
    mBufferQueue->setDefaultBufferFormat(mHwc.getFormat(disp));
    mBufferQueue->setDefaultBufferSize(mHwc.getWidth(disp),  mHwc.getHeight(disp));
    mBufferQueue->setSynchronousMode(true);
    mBufferQueue->setDefaultMaxBufferCount(NUM_FRAMEBUFFER_SURFACE_BUFFERS);
}
```

Create a GraphicBufferAlloc instance

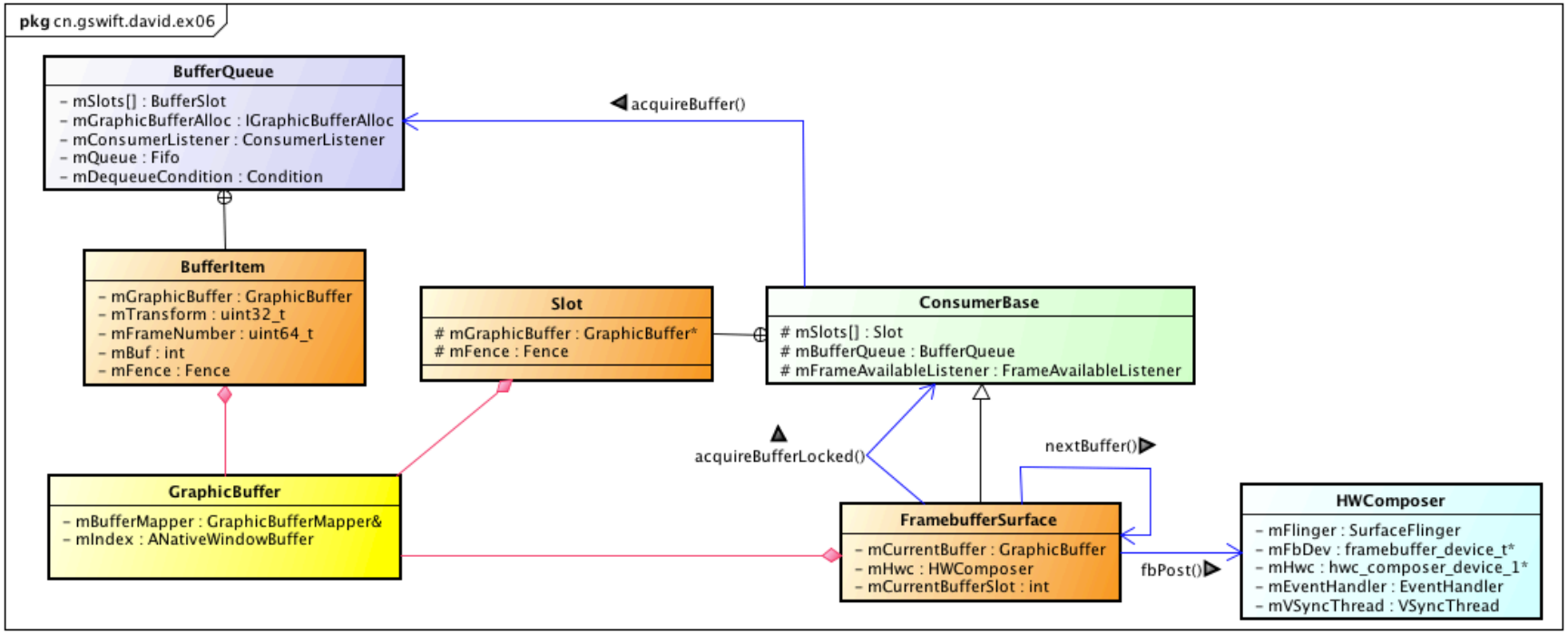
Create a BufferQueue instance

```
BufferQueue::BufferQueue(bool allowSynchronousMode,
        const sp<IGraphicBufferAlloc>& allocator) :
    mDefaultWidth(1),
    mDefaultHeight(1),
    mMaxAcquiredBufferCount(1),
    mDefaultMaxBufferCount(2),
    mOverrideMaxBufferCount(0),
    mSynchronousMode(false),
    mAllowSynchronousMode(allowSynchronousMode),
    mConnectedApi(NO_CONNECTED_API),
    mAbandoned(false),
    mFrameCounter(0),
    mBufferHasBeenQueued(false),
    mDefaultBufferFormat(PIXEL_FORMAT_RGBA_8888),
    mConsumerUsageBits(0),
    mTransformHint(0)
{
    // Choose a name using the PID and a process-unique ID.
    mConsumerName = String8::format("unnamed-%d-%d", getpid(), createProcessUniqueId());

    ST_LOGV("BufferQueue");
    if (allocator == NULL) {
        sp<ISurfaceComposer> composer(ComposerService::getComposerService());
        mGraphicBufferAlloc = composer->createGraphicBufferAlloc();
        if (mGraphicBufferAlloc == 0) {
            ST_LOGE("createGraphicBufferAlloc() failed in BufferQueue()");
        }
    } else {
        mGraphicBufferAlloc = allocator;
    }
}
```

IGraphicBufferAlloc

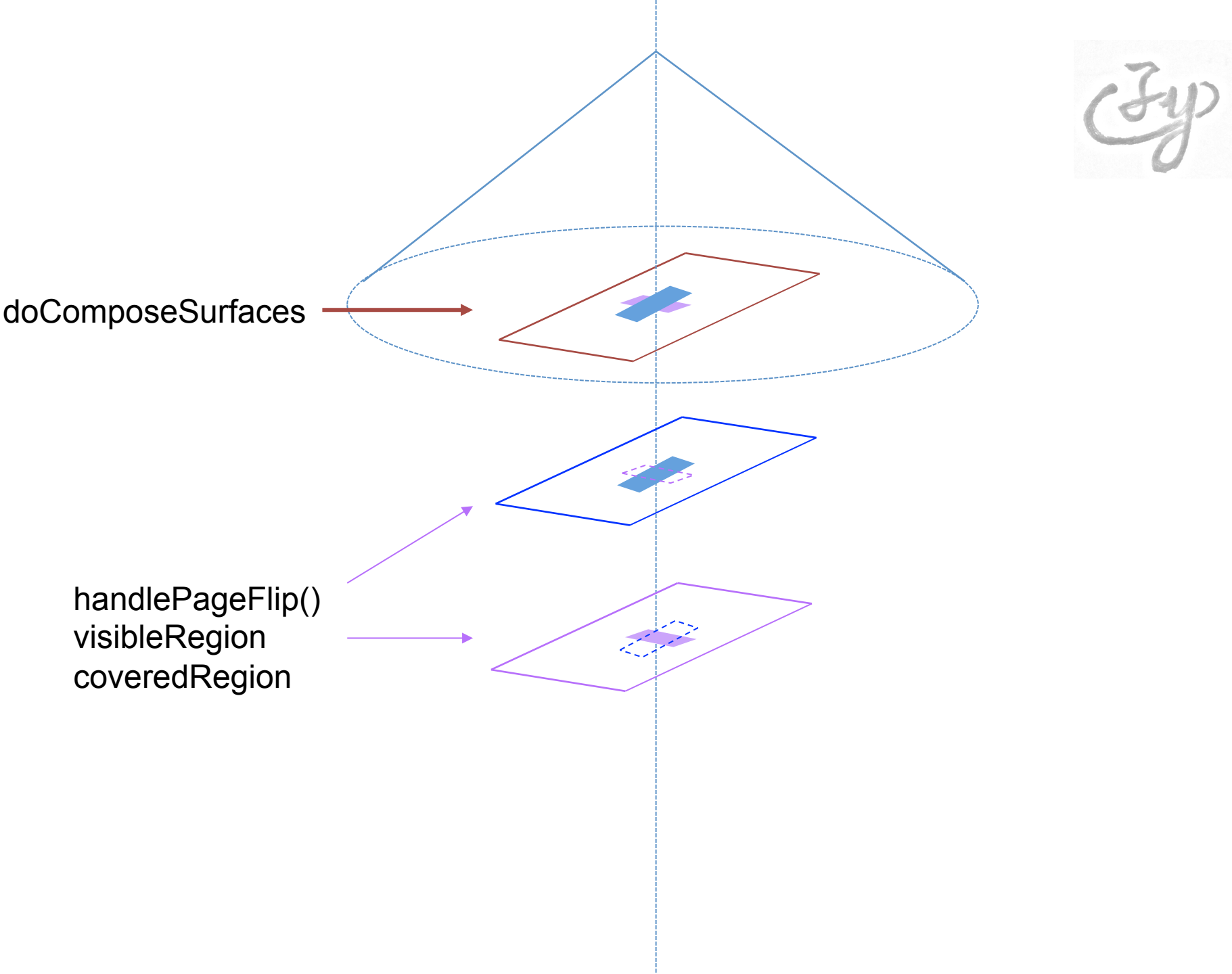


fbPost()

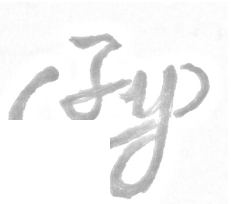

pkg cn.gswift.david.ex06
BufferQueue
- mSlots[] : BufferSlot
- mGraphicBufferAlloc : IGraphicBufferAlloc
- mConsumerListener : ConsumerListener
- mQueue : Fifo
- mDequeueCondition : Condition
acquireBuffer()
BufferItem
- mGraphicBuffer : GraphicBuffer
- mTransform : uint32_t
- mFrameNumber : uint64_t
- mBuf : int
- mFence : Fence
Slot
mGraphicBuffer : GraphicBuffer*
mFence : Fence
ConsumerBase
# mSlots[] : Slot
# mBufferQueue : BufferQueue
# mFrameAvailableListener : FrameAvailableListener
acquireBufferLocked()
nextBuffer()
GraphicBuffer
- mBufferMapper : GraphicBufferMapper&
- mIndex : ANativeWindowBuffer
FramebufferSurface
- mCurrentBuffer : GraphicBuffer
- mHwc : HWComposer
- mCurrentBufferSlot : int
fbPost()
HWComposer
- mFlinger : SurfaceFlinger
- mFbDev : framebuffer_device_t*
- mHwc : hwc_composer_device_1*
- mEventHandler : EventHandler
- mVSyncThread : VSyncThread

doComposeSurfaces

SurfaceFlinger

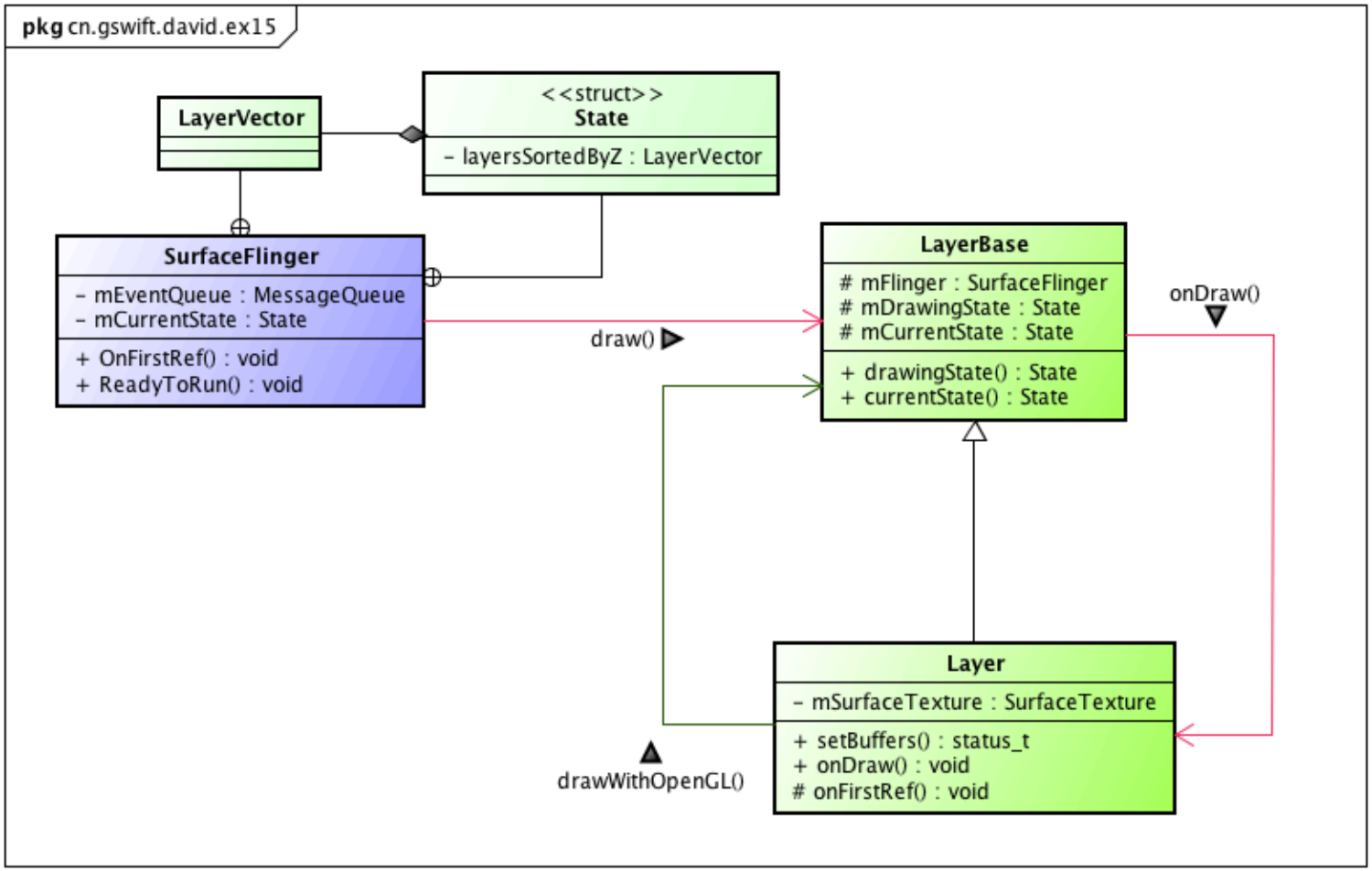

doComposeSurfaces
handlePageFlip()
visibleRegion
coveredRegion

SurfaceFlinger::doComposeSurfaces



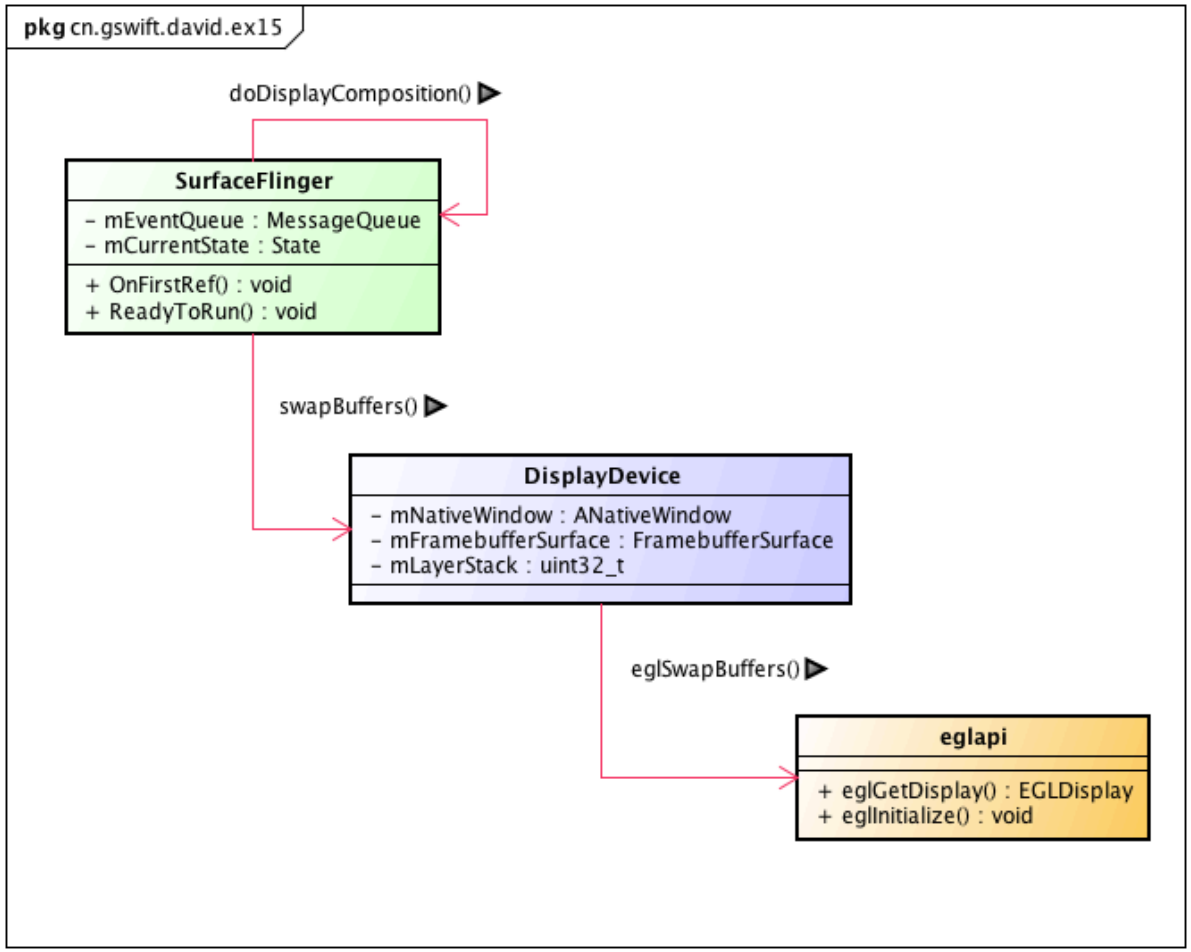
```
const Vector< sp<LayerBase> >& layers(hw->getVisibleLayersSortedByZ());
const size_t count = layers.size();
const Transform& tr = hw->getTransform();
if (cur != end) {
    // we're using h/w composer
    for (size_t i=0 ; i<count && cur!=end ; ++i, ++cur) {
        const sp<LayerBase>& layer(layers[i]);
        const Region clip(dirty.intersect(tr.transform(layer->visibleRegion)));
        if (!clip.isEmpty()) {
            switch (cur->getCompositionType()) {
                case HWC_OVERLAY: {
                    if ((cur->getHints() & HWC_HINT_CLEAR_FB)
                            && i
                            && layer->isOpaque()
                            && hasGlesComposition) {
                        // never clear the very first layer since we're
                        // guaranteed the FB is already cleared
                        layer->clearWithOpenGL(hw, clip);
                    }
                    break;
                }
                case HWC_FRAMEBUFFER: {
                    layer->draw(hw, clip);
                    break;
                }
                case HWC_FRAMEBUFFER_TARGET: {
                    // this should not happen as the iterator shouldn't
                    // let us get there.
                    ALOGW("HWC_FRAMEBUFFER_TARGET found in hwc list (index=%d)", i);
                    break;
                }
            }
        }
        layer->setAcquireFence(hw, *cur);
```

Draw layer with OpenGL



```
void SurfaceFlinger::doDisplayComposition(const sp<const DisplayDevice>& hw,
        const Region& inDirtyRegion)
{
    Region dirtyRegion(inDirtyRegion);

    // compute the invalid region
    hw->swapRegion.orSelf(dirtyRegion);

    uint32_t flags = hw->getFlags();
    if (flags & DisplayDevice::SWAP_RECTANGLE) {
        // we can redraw only what's dirty, but since SWAP_RECTANGLE only
        // takes a rectangle, we must make sure to update that whole
        // rectangle in that case
        dirtyRegion.set(hw->swapRegion.bounds());
    } else {
        if (flags & DisplayDevice::PARTIAL_UPDATES) {
            // We need to redraw the rectangle that will be updated
            // (pushed to the framebuffer).
            // This is needed because PARTIAL_UPDATES only takes one
            // rectangle instead of a region (see DisplayDevice::flip())
            dirtyRegion.set(hw->swapRegion.bounds());
        } else {
            // we need to redraw everything (the whole screen)
            dirtyRegion.set(hw->bounds());
            hw->swapRegion = dirtyRegion;
        }
    }

    doComposeSurfaces(hw, dirtyRegion);

    // update the swap region and clear the dirty region
    hw->swapRegion.orSelf(dirtyRegion);

    // swap buffers (presentation)
    hw->swapBuffers(getHwComposer());
}
```

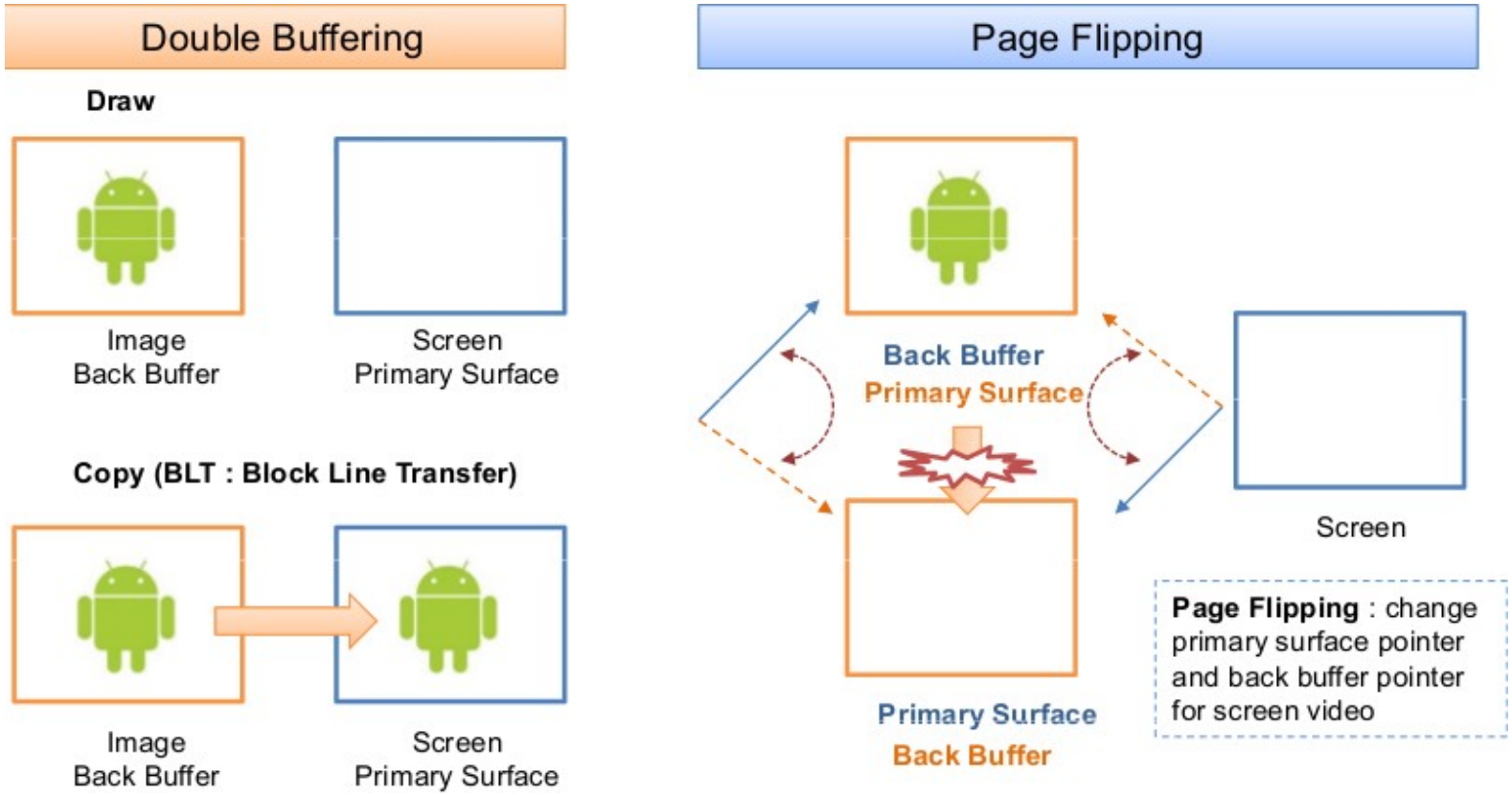
Swap Buffers



handlePageFlip

SurfaceFlinger

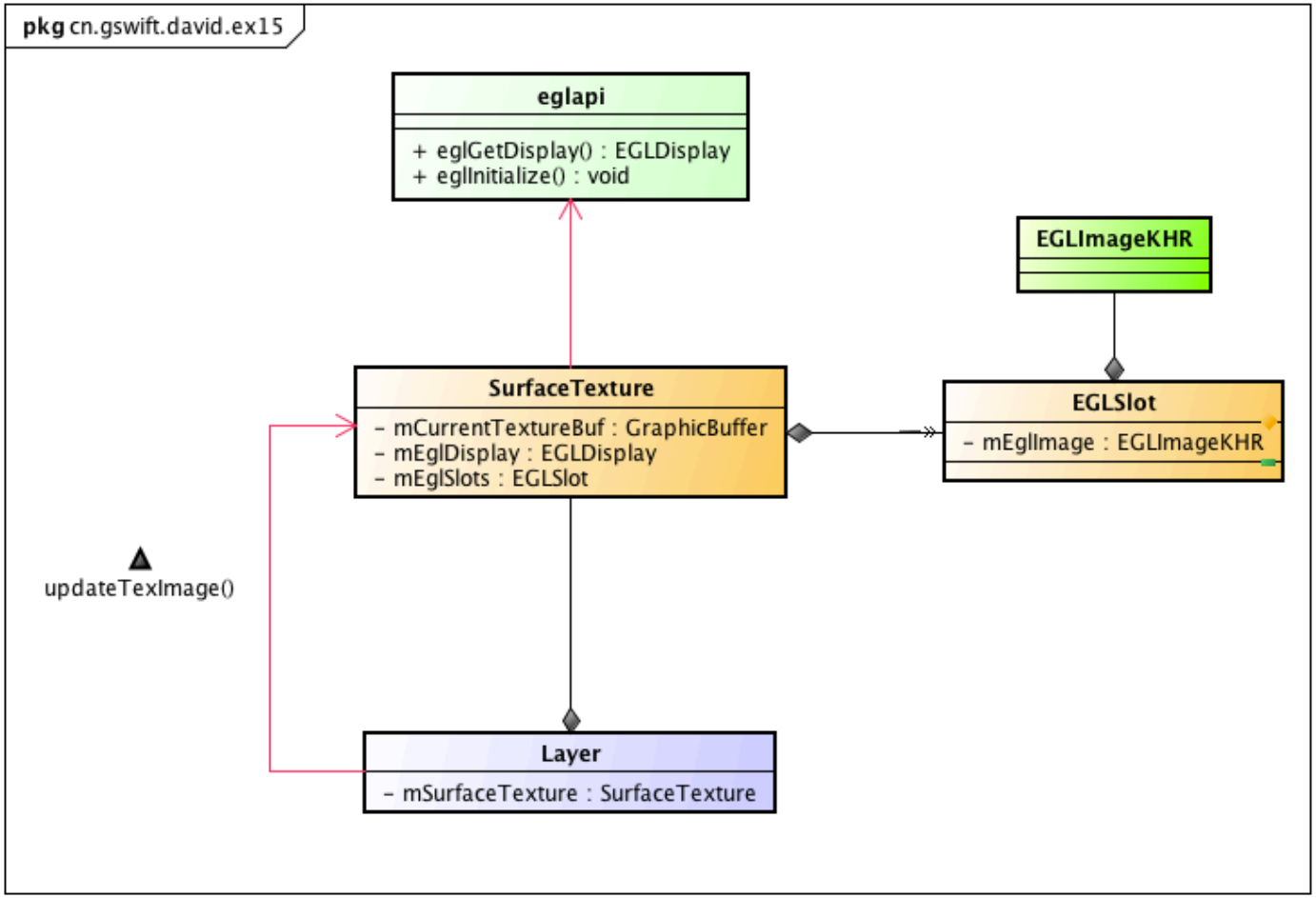
Page Flipping



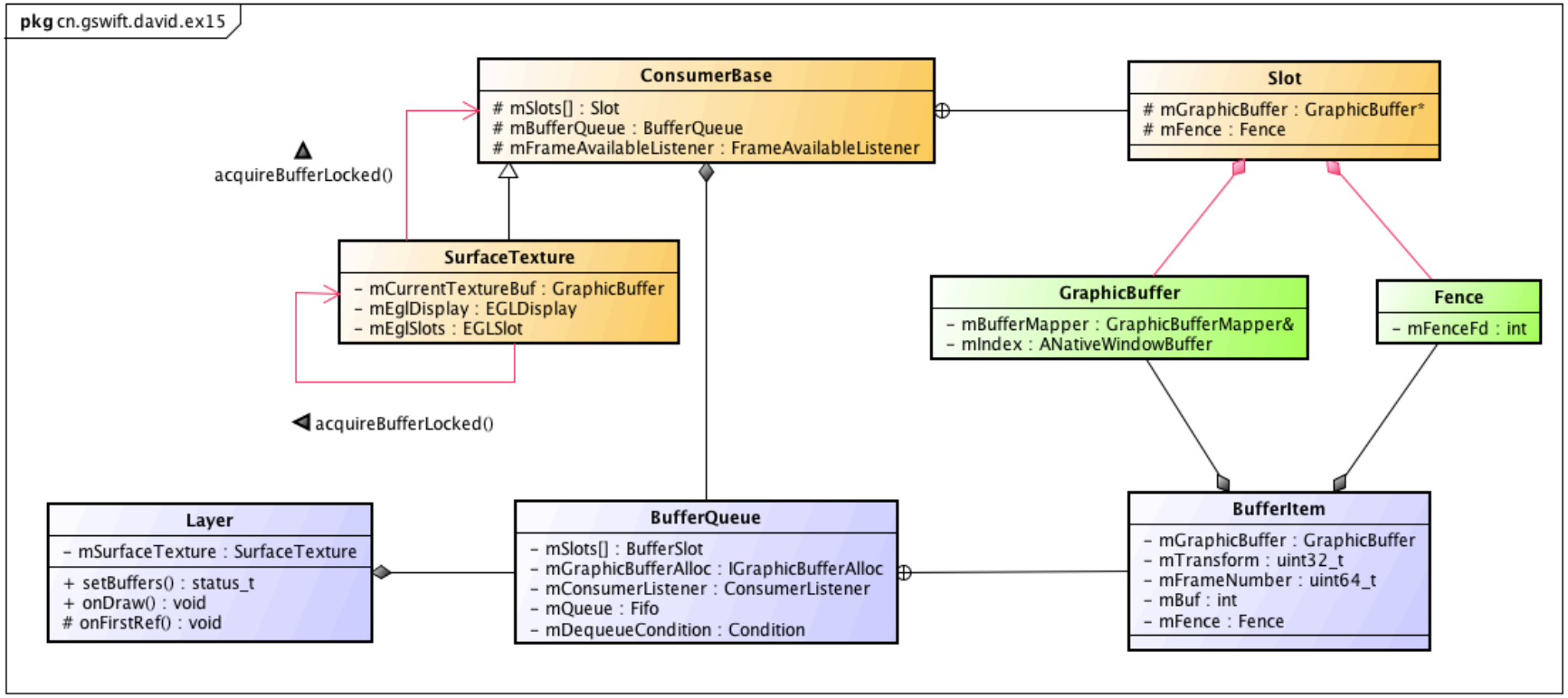
Source from: http://0xlab.org/

```
void SurfaceFlinger::handlePageFlip()
{
    Region dirtyRegion;

    bool visibleRegions = false;
    const LayerVector& currentLayers(mDrawingState.layersSortedByZ);
    const size_t count = currentLayers.size();
    for (size_t i=0 ; i<count ; i++) {
        const sp<LayerBase>& layer(currentLayers[i]);
        const Region dirty(layer->latchBuffer(visibleRegions));
        const Layer::State& s(layer->drawingState());
        invalidateLayerStack(s.layerStack, dirty);
    }

    mVisibleRegionsDirty |= visibleRegions;
}
```

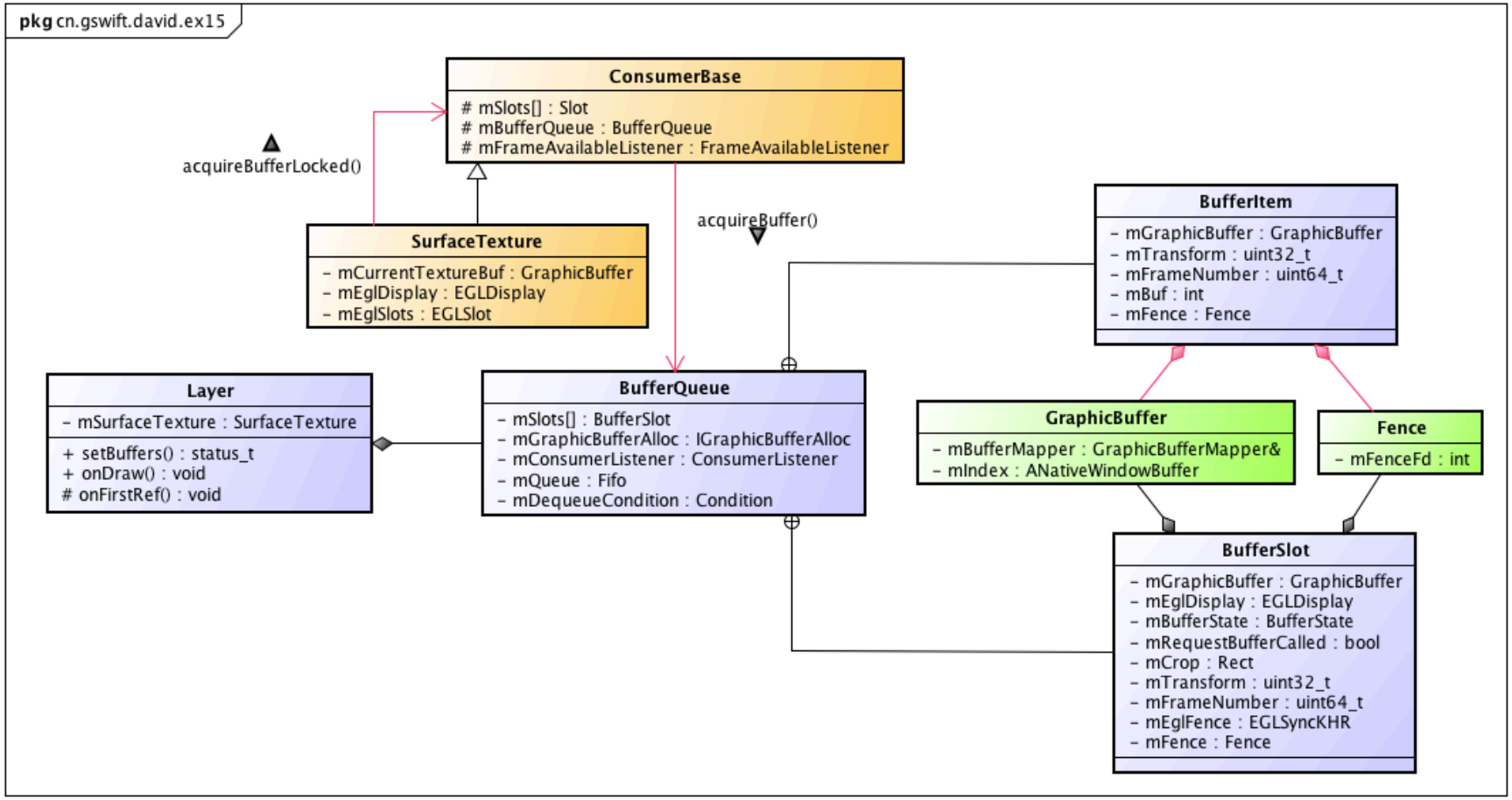
latchBuffer()->updatetexImage()



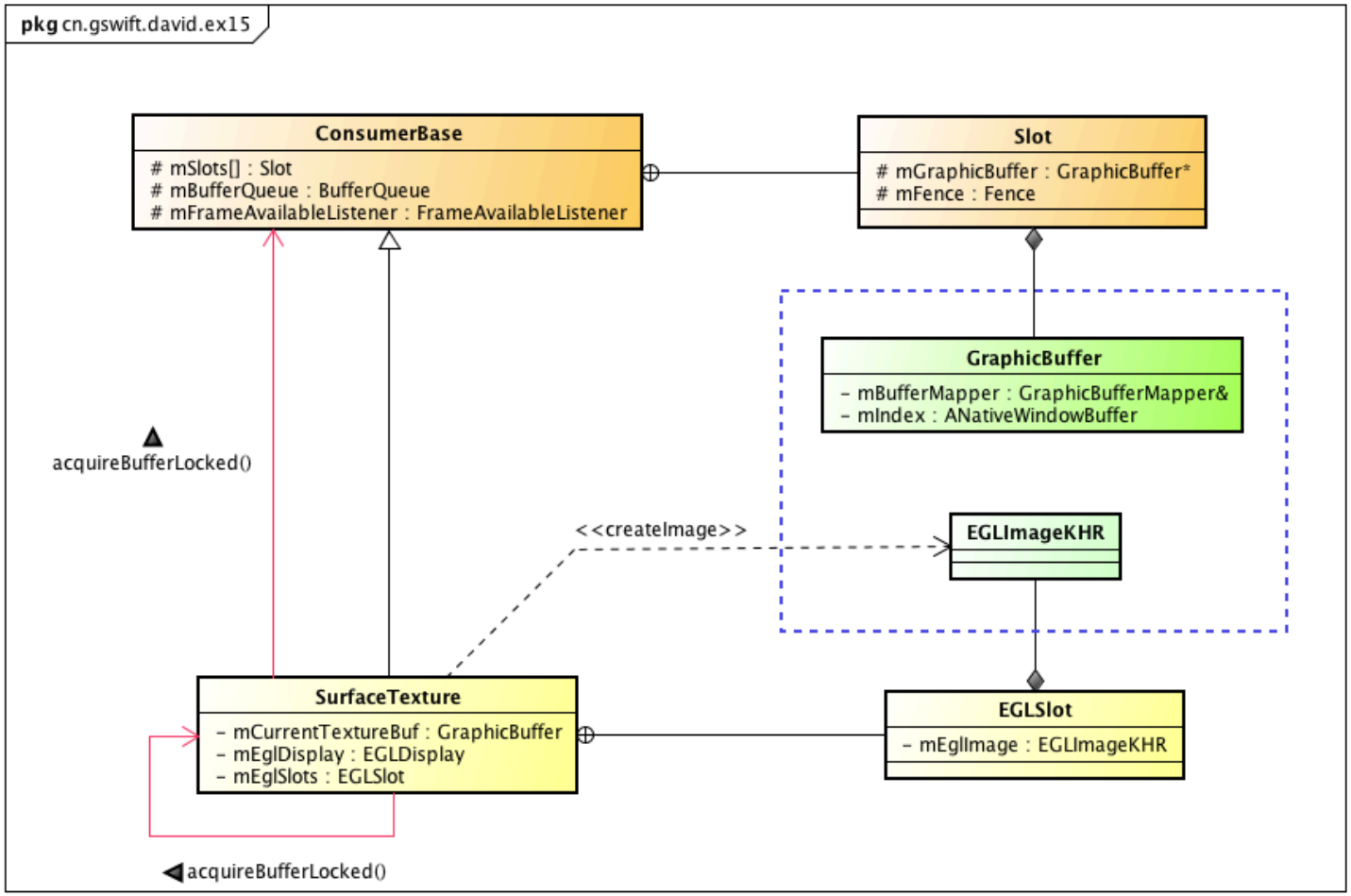
acquireBufferLocked(1)



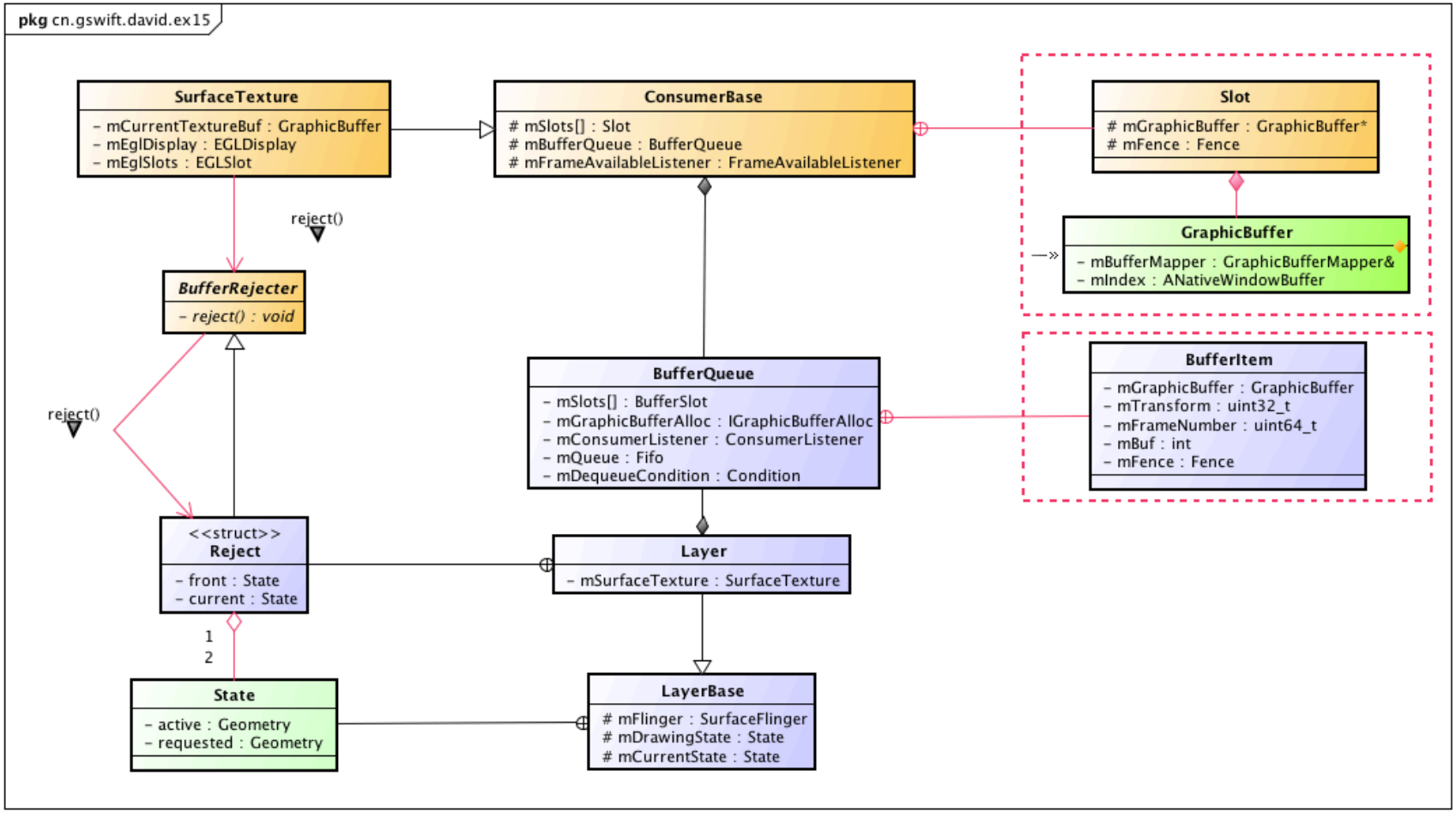
acquireBuffer(1)



acquireBufferLocked()



Reject the GraphicBuffer



Layer.cpp

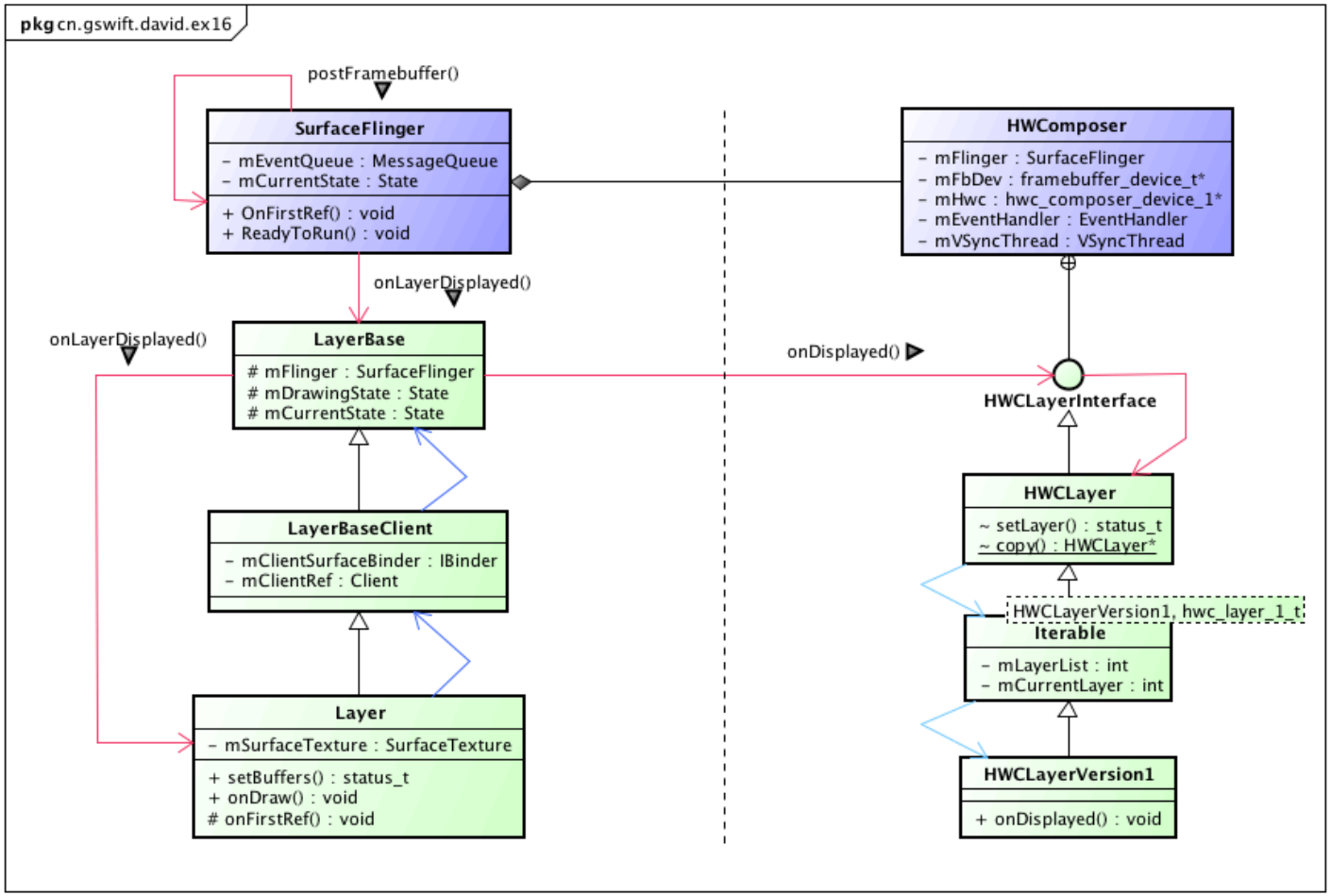
```
Reject r(mDrawingState, currentState(), recomputeVisibleRegions);

if (mSurfaceTexture->updateTexImage(&r, true) < NO_ERROR) {
    // something happened!
    recomputeVisibleRegions = true;
    return outDirtyRegion;
}
```

```
   struct Reject : public SurfaceTexture::BufferRejecter {
       Layer::State& front;
       Layer::State& current;
       bool& recomputeVisibleRegions;
       Reject(Layer::State& front, Layer::State& current,
              bool& recomputeVisibleRegions)
           : front(front), current(current),
             recomputeVisibleRegions(recomputeVisibleRegions) {
       }
```

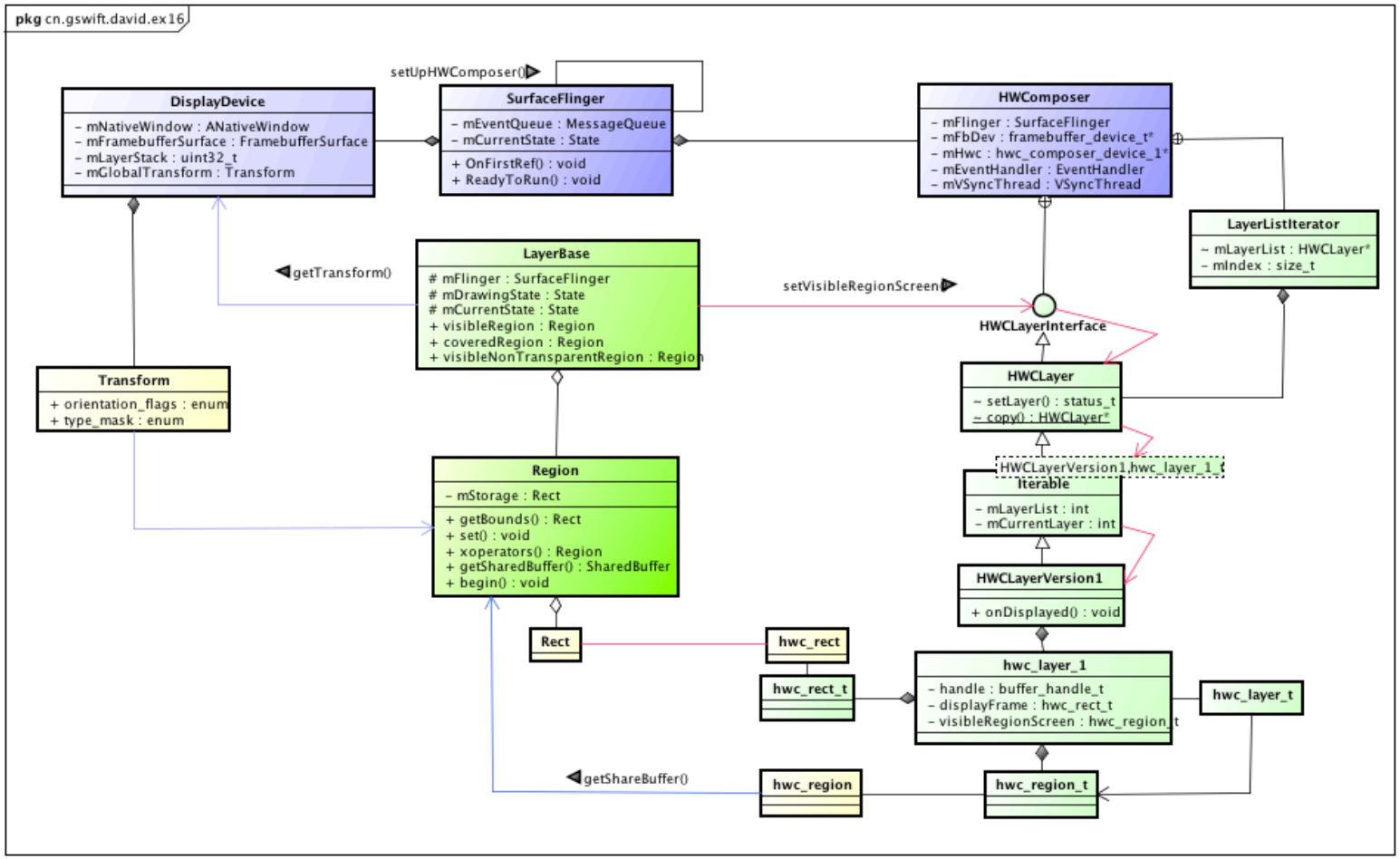
PostFramebuffer

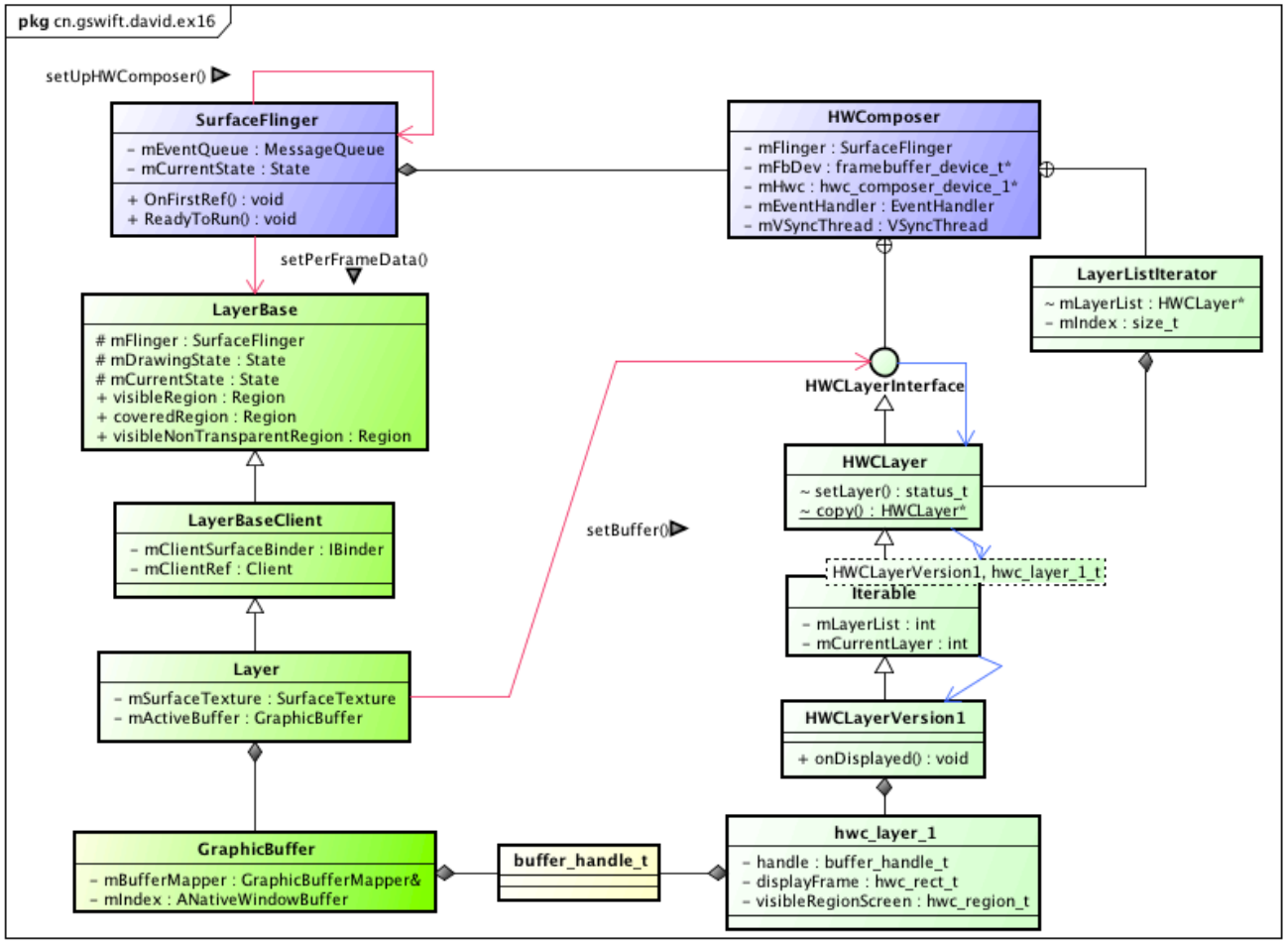
SurfaceFlinger

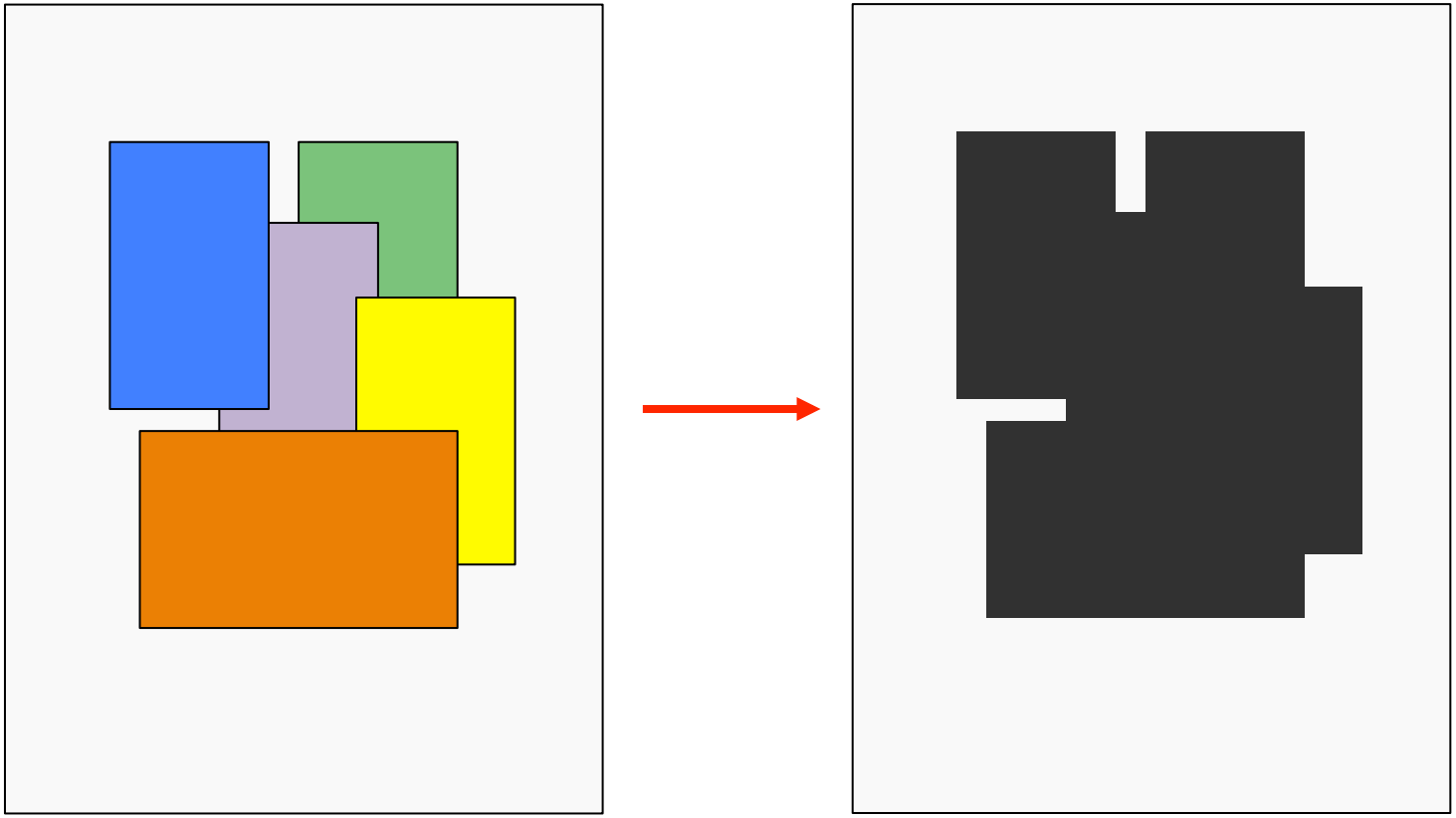
```
void SurfaceFlinger::handleMessageTransaction() {
    uint32_t transactionFlags = peekTransactionFlags(eTransactionMask);
    if (transactionFlags) {
        handleTransaction(transactionFlags);
    }
}

void SurfaceFlinger::handleMessageInvalidate() {
    ATRACE_CALL();
    handlePageFlip();
}

void SurfaceFlinger::handleMessageRefresh() {
    ATRACE_CALL();
    preComposition();
    rebuildLayerStacks();
    setUpHWComposer();
    doDebugFlashRegions();
    doComposition();
    postComposition();
}
```


pkg cn.gswift.david.ex16
postFramebuffer()
SurfaceFlinger
- mEventQueue : MessageQueue
- mCurrentState : State
+ OnFirstRef() : void
+ ReadyToRun() : void
HWComposer
- mFlinger : SurfaceFlinger
- mFbDev : framebuffer_device_t*
- mHwc : hwc_composer_device_1*
- mEventHandler : EventHandler
- mVSyncThread : VSyncThread
onLayerDisplayed()
onLayerDisplayed()
LayerBase
# mFlinger : SurfaceFlinger
# mDrawingState : State
# mCurrentState : State
onDisplayed()
HWCLayerInterface
HWCLayer
~ setLayer() : status_t
~ copy() : HWCLayer*
LayerBaseClient
- mClientSurfaceBinder : IBinder
- mClientRef : Client
HWCLayerVersion1, hwc_layer_1_t
Iterable
- mLayerList : int
- mCurrentLayer : int
Layer
- mSurfaceTexture : SurfaceTexture
+ setBuffers() : status_t
+ onDraw() : void
# onFirstRef() : void
HWCLayerVersion1
+ onDisplayed() : void

```
virtual void setVisibleRegionScreen(const Region& reg) {
    // Region::getSharedBuffer creates a reference to the underlying
    // SharedBuffer of this Region, this reference is freed
    // in onDisplayed()
    hwc_region_t& visibleRegion = getLayer()->visibleRegionScreen;
    SharedBuffer const* sb = reg.getSharedBuffer(&visibleRegion.numRects);
    visibleRegion.rects = reinterpret_cast<hwc_rect_t const *>(sb->data());
}
virtual void setBuffer(const sp<GraphicBuffer>& buffer) {
    if (buffer == 0 || buffer->handle == 0) {
        getLayer()->compositionType = HWC_FRAMEBUFFER;
        getLayer()->flags |= HWC_SKIP_LAYER;
        getLayer()->handle = 0;
    } else {
        getLayer()->handle = buffer->handle;
    }
}
virtual void onDisplayed() {
    hwc_region_t& visibleRegion = getLayer()->visibleRegionScreen;
    SharedBuffer const* sb = SharedBuffer::bufferFromData(visibleRegion.rects);
    if (sb) {
        sb->release();
        // not technically needed but safer
        visibleRegion.numRects = 0;
        visibleRegion.rects = NULL;
    }

    getLayer()->acquireFenceFd = -1;
}
```


pkg cn.gswift.david.ex16
setUpHWComposer()
DisplayDevice
- mNativeWindow : ANativeWindow
- mFramebufferSurface : FramebufferSurface
- mLayerStack : uint32_t
- mGlobalTransform : Transform
SurfaceFlinger
- mEventQueue : MessageQueue
- mCurrentState : State
+ OnFirstRef() : void
+ ReadyToRun() : void
HWComposer
- mFlinger : SurfaceFlinger
- mFbDev : framebuffer_device_t*
- mHwc : hwc_composer_device_1*
- mEventHandler : EventHandler
- mVSyncThread : VSyncThread
LayerListIterator
~ mLayerList : HWCLayer*
- mIndex : size_t
getTransform()
LayerBase
mFlinger : SurfaceFlinger
mDrawingState : State
mCurrentState : State
+ visibleRegion : Region
+ coveredRegion : Region
+ visibleNonTransparentRegion : Region
setVisibleRegionScreen
HWCLayerInterface
Transform
+ orientation_flags : enum
+ type_mask : enum
HWCLayer
~ setLayer() : status_t
~ copy() : HWCLayer*
HWCLayerVersion1,hwc_layer_1_t
Iterable
- mLayerList : int
- mCurrentLayer : int
Region
- mStorage : Rect
+ getBounds() : Rect
+ set() : void
+ xoperators() : Region
+ getSharedBuffer() : SharedBuffer
+ begin() : void
HWCLayerVersion1
+ onDisplayed() : void
Rect
hwc_rect
hwc_rect_t
hwc_layer_1
- handle : buffer_handle_t
- displayFrame : hwc_rect_t
- visibleRegionScreen : hwc_region_t
hwc_layer_t
getShareBuffer()
hwc_region
hwc_region_t

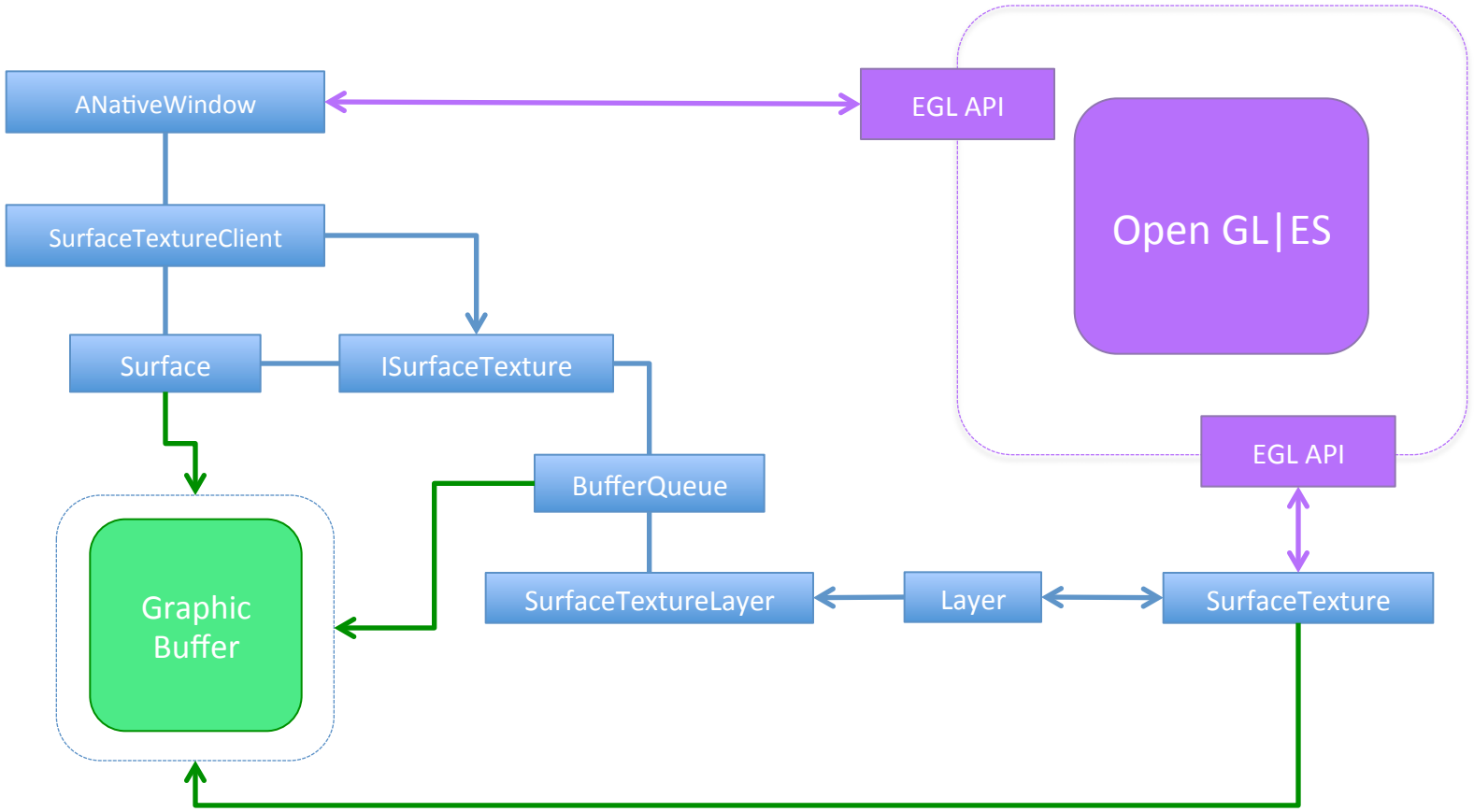

pkg cn.gswift.david.ex16
setUpHWComposer()
SurfaceFlinger
- mEventQueue : MessageQueue
- mCurrentState : State
+ OnFirstRef() : void
+ ReadyToRun() : void
setPerFrameData()
HWComposer
- mFlinger : SurfaceFlinger
- mFbDev : framebuffer_device_t*
- mHwc : hwc_composer_device_1*
- mEventHandler : EventHandler
- mVSyncThread : VSyncThread
LayerListIterator
~ mLayerList : HWCLayer*
- mIndex : size_t
LayerBase
mFlinger : SurfaceFlinger
mDrawingState : State
mCurrentState : State
+ visibleRegion : Region
+ coveredRegion : Region
+ visibleNonTransparentRegion : Region
HWCLayerInterface
HWCLayer
~ setLayer() : status_t
~ copy() : HWCLayer*
LayerBaseClient
- mClientSurfaceBinder : IBinder
- mClientRef : Client
setBuffer()
HWCLayerVersion1, hwc_layer_1_t
Iterable
- mLayerList : int
- mCurrentLayer : int
Layer
- mSurfaceTexture : SurfaceTexture
- mActiveBuffer : GraphicBuffer
HWCLayerVersion1
+ onDisplayed() : void
GraphicBuffer
- mBufferMapper : GraphicBufferMapper&
- mIndex : ANativeWindowBuffer
buffer_handle_t
hwc_layer_1
- handle : buffer_handle_t
- displayFrame : hwc_rect_t
- visibleRegionScreen : hwc_region_t

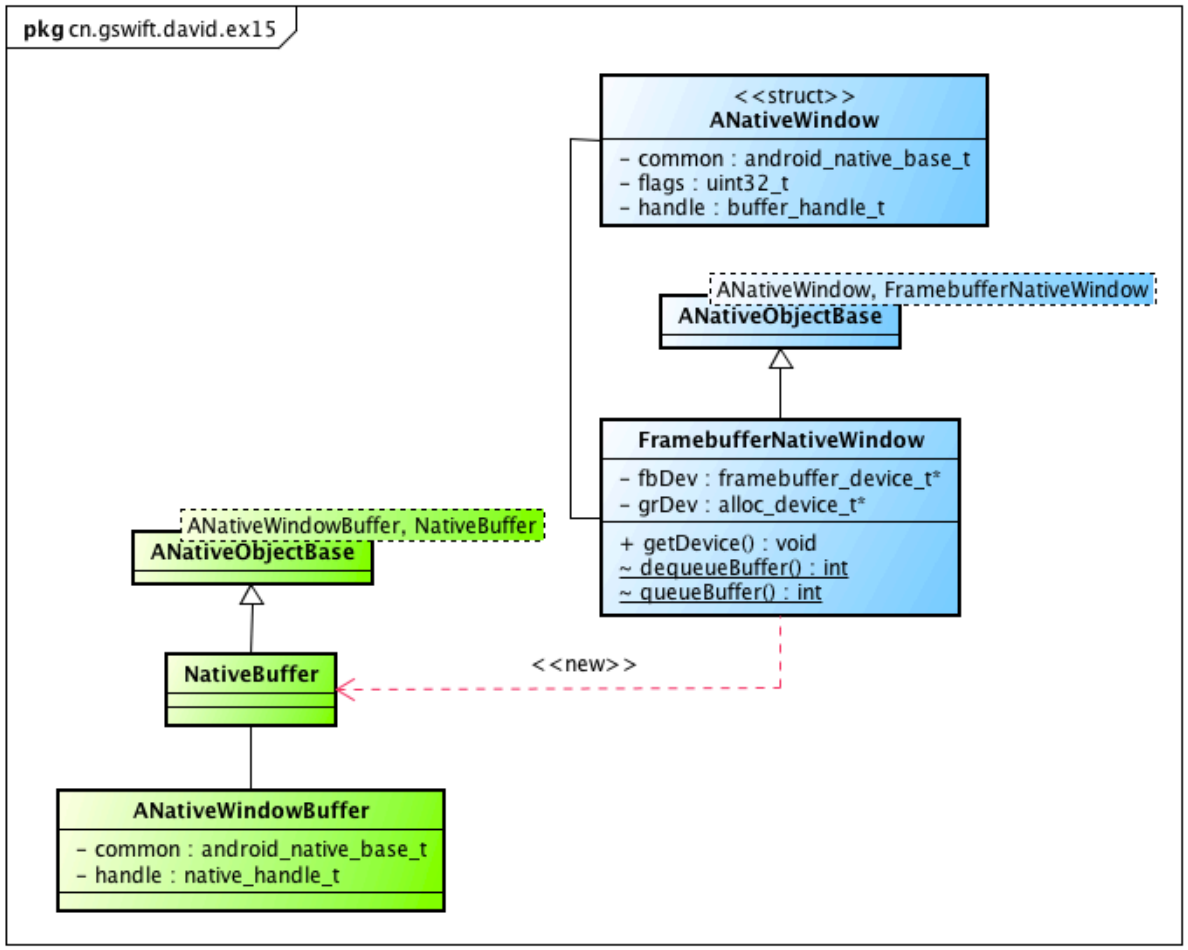


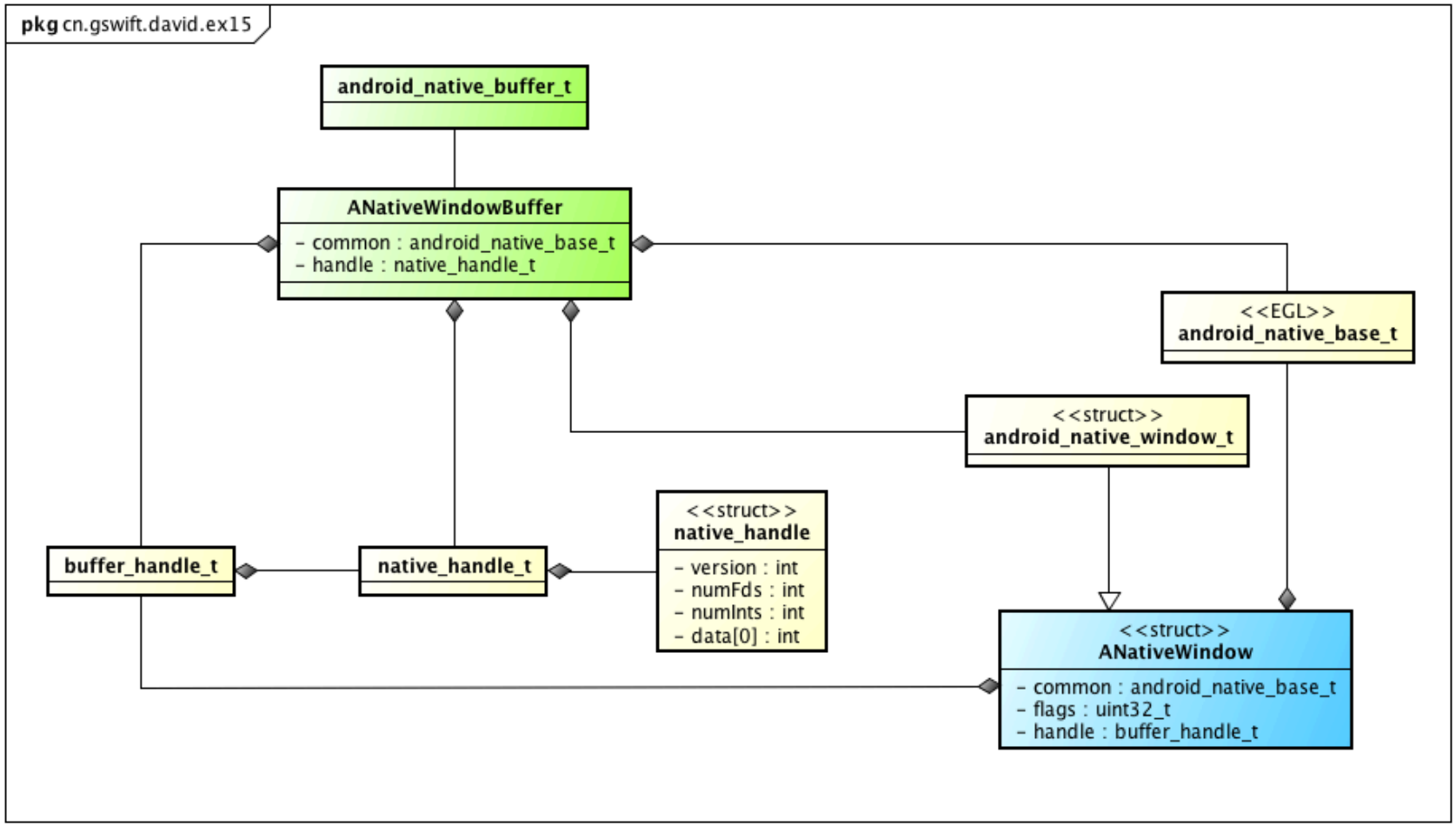
OPENGL | ES Library , Graphic Buffer

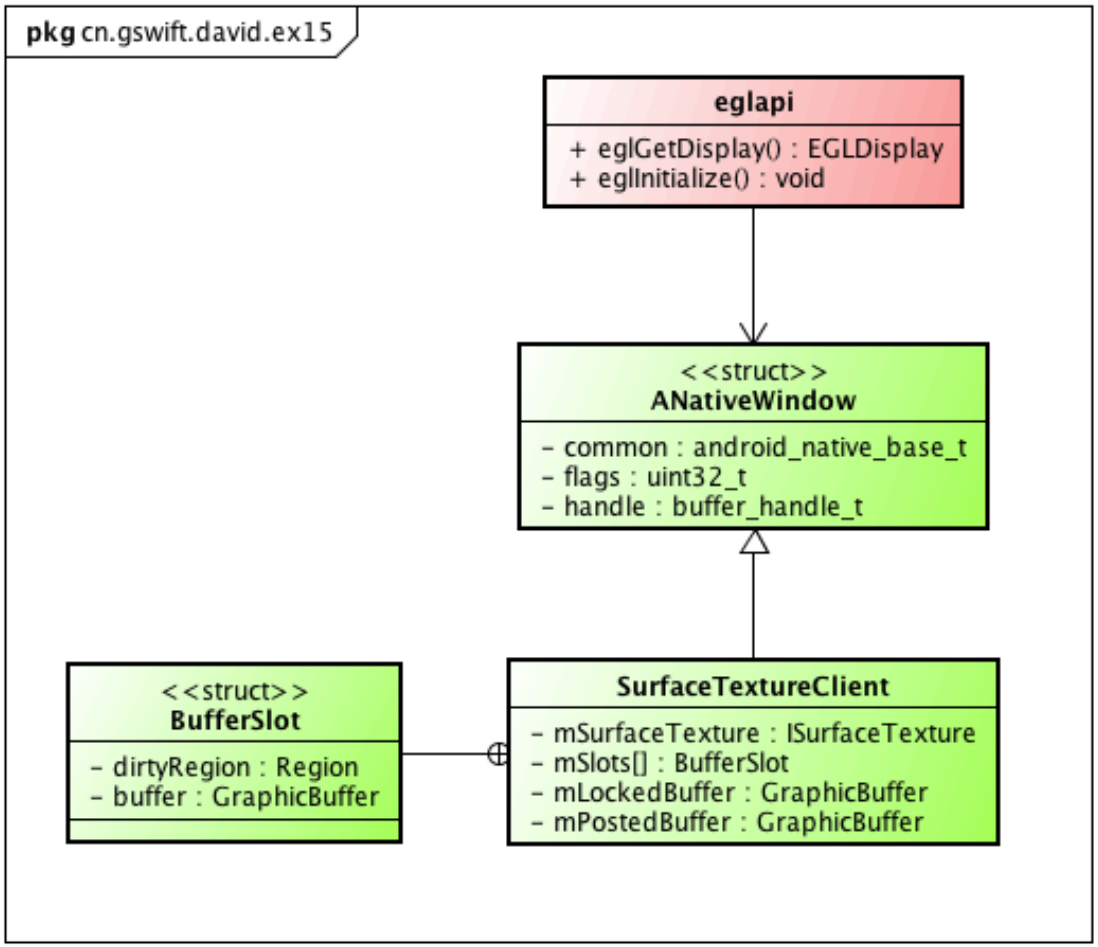
SurfaceFlinger

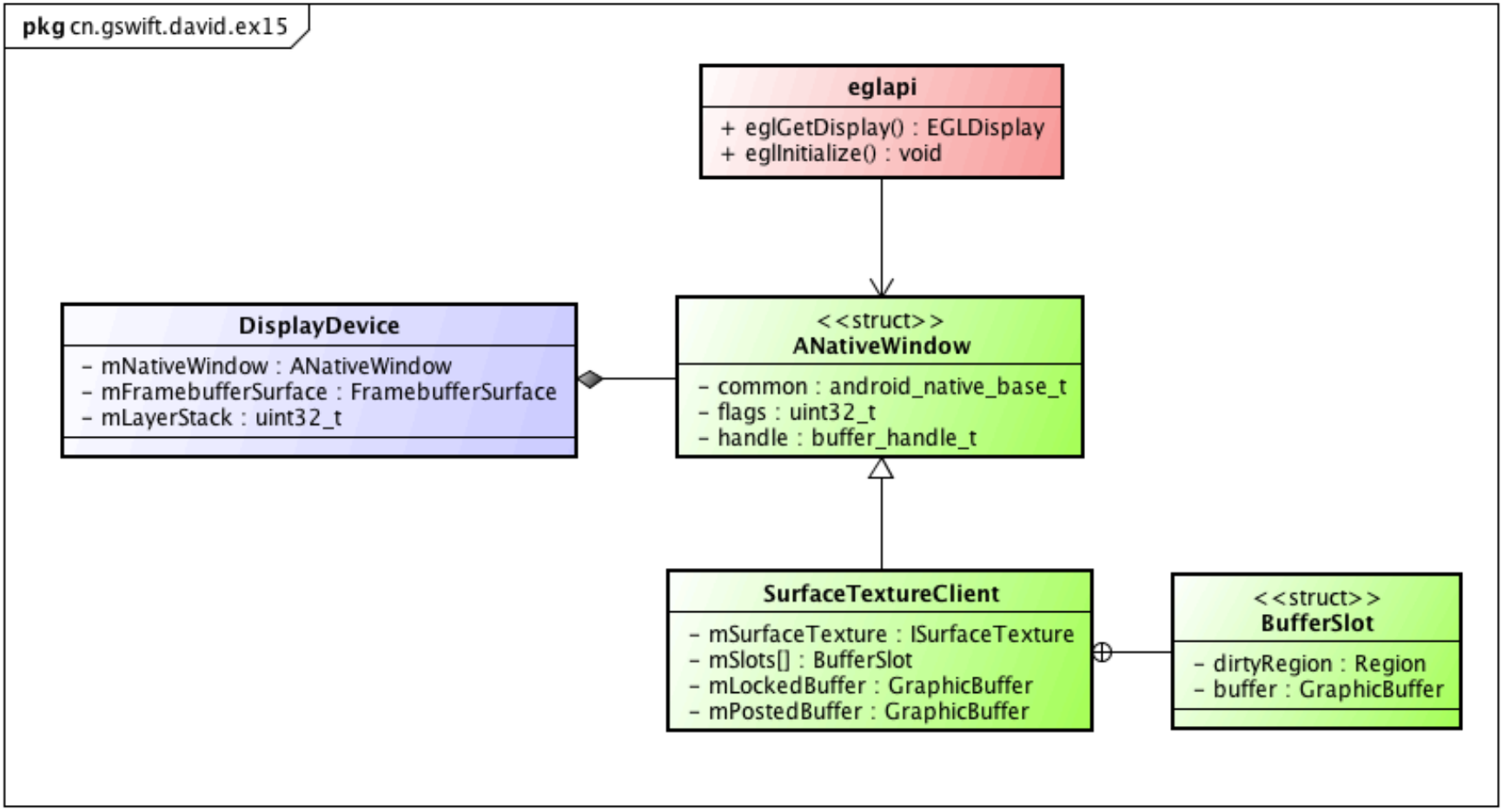
SurfaceFlinger & OpenGL&GraphicBuffer



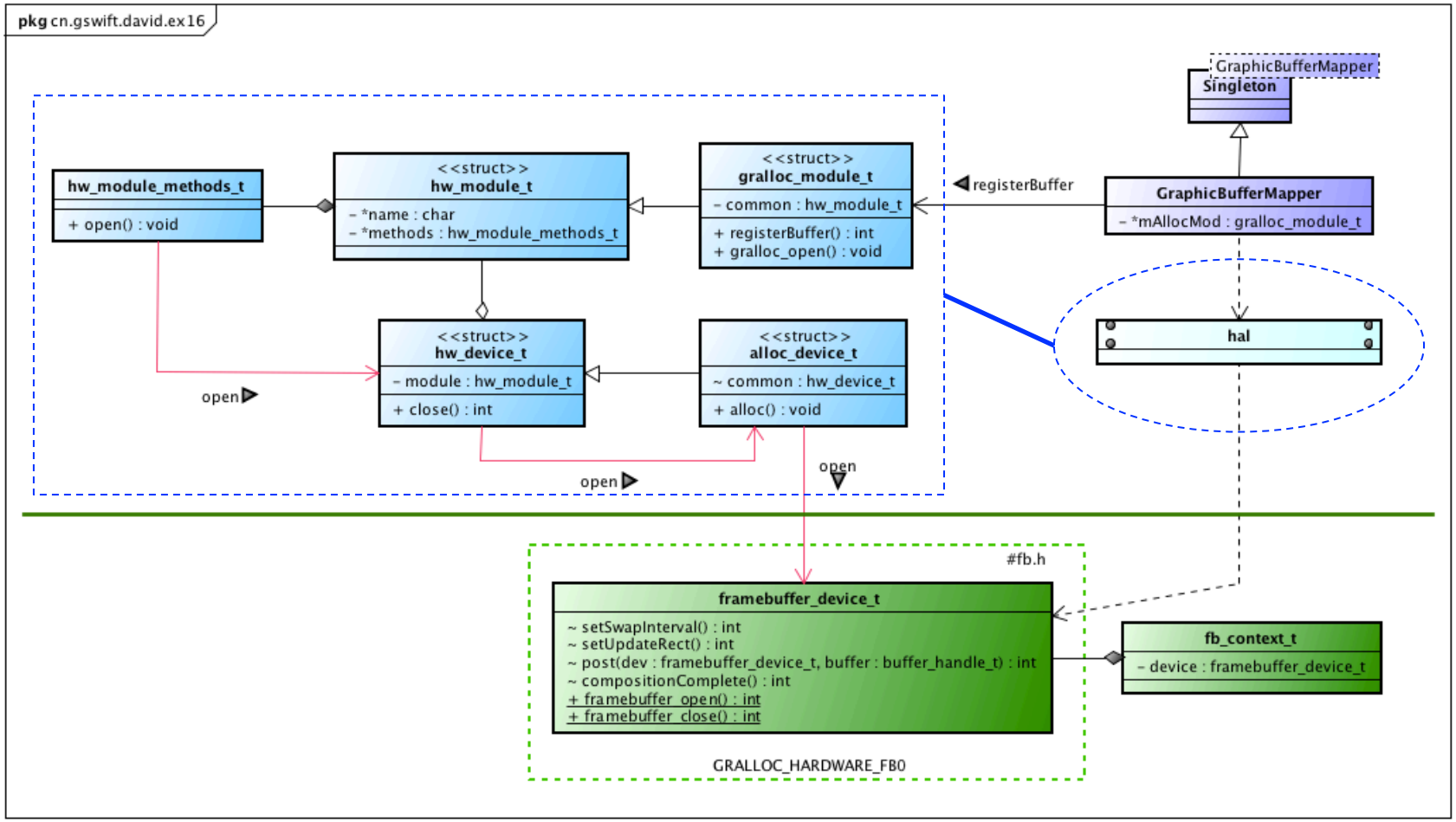

pkg cn.gswift.david.ex15
<<struct>>
ANativeWindow
- common : android_native_base_t
- flags : uint32_t
- handle : buffer_handle_t
ANativeWindow, FramebufferNativeWindow
ANativeObjectBase
FramebufferNativeWindow
- fbDev : framebuffer_device_t*
- grDev : alloc_device_t*
+ getDevice() : void
~ dequeueBuffer() : int
~ queueBuffer() : int
ANativeWindowBuffer, NativeBuffer
ANativeObjectBase
<<new>>
NativeBuffer
ANativeWindowBuffer
- common : android_native_base_t
- handle : native_handle_t


pkg cn.gswift.david.ex15
android_native_buffer_t
ANativeWindowBuffer
- common : android_native_base_t
- handle : native_handle_t
<<EGL>>
android_native_base_t
<<struct>>
android_native_window_t
<<struct>>
native_handle
- version : int
- numFds : int
- numInts : int
- data[0] : int
buffer_handle_t
native_handle_t
<<struct>>
ANativeWindow
- common : android_native_base_t
- flags : uint32_t
- handle : buffer_handle_t

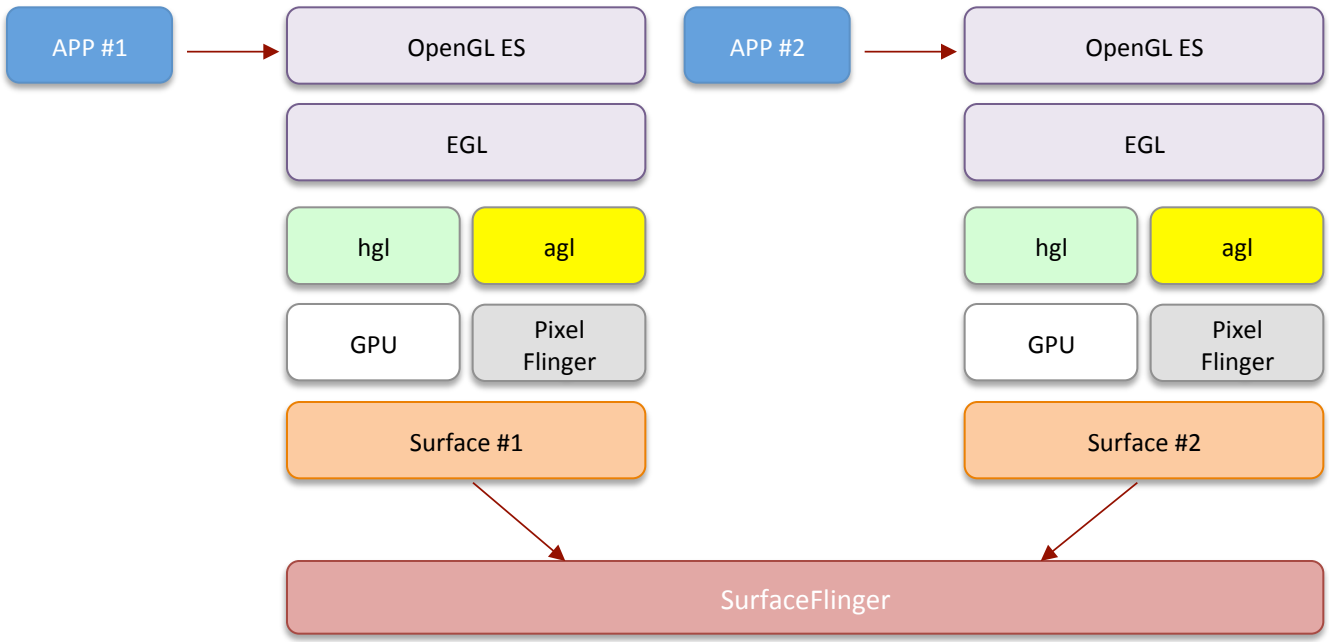

pkg cn.gswift.david.ex15
eglapi
+ eglGetDisplay() : EGLDisplay
+ eglInitialize() : void
<<struct>>
ANativeWindow
- common : android_native_base_t
- flags : uint32_t
- handle : buffer_handle_t
<<struct>>
BufferSlot
- dirtyRegion : Region
- buffer : GraphicBuffer
SurfaceTextureClient
- mSurfaceTexture : ISurfaceTexture
- mSlots[] : BufferSlot
- mLockedBuffer : GraphicBuffer
- mPostedBuffer : GraphicBuffer


pkg cn.gswift.david.ex15
eglapi
+ eglGetDisplay() : EGLDisplay
+ eglInitialize() : void
DisplayDevice
- mNativeWindow : ANativeWindow
- mFramebufferSurface : FramebufferSurface
- mLayerStack : uint32_t
<<struct>>
ANativeWindow
- common : android_native_base_t
- flags : uint32_t
- handle : buffer_handle_t
SurfaceTextureClient
- mSurfaceTexture : ISurfaceTexture
- mSlots[] : BufferSlot
- mLockedBuffer : GraphicBuffer
- mPostedBuffer : GraphicBuffer
<<struct>>
BufferSlot
- dirtyRegion : Region
- buffer : GraphicBuffer

framebuffer_device_t



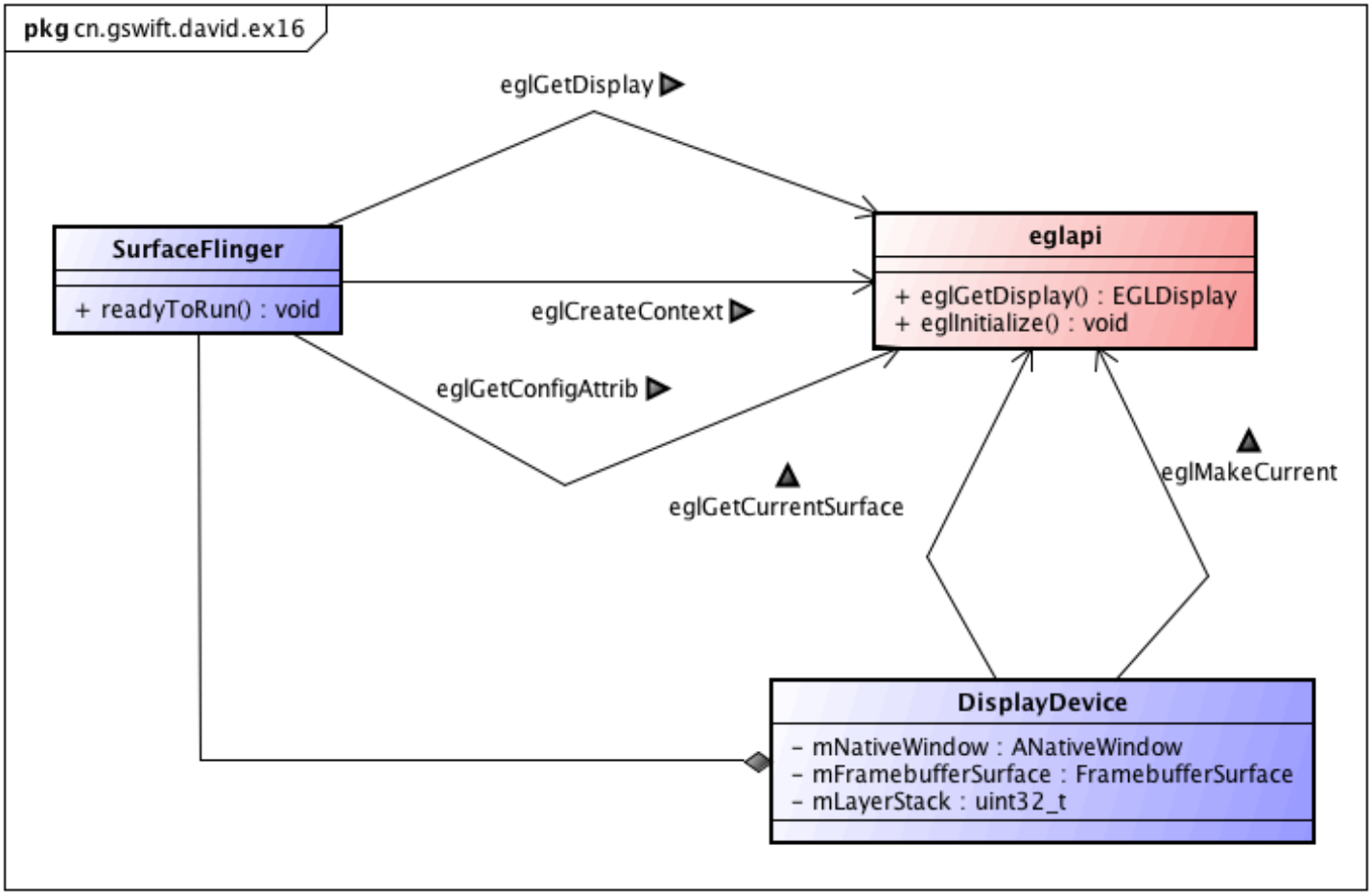
HW: From OpenGL To SurfaceFlinger



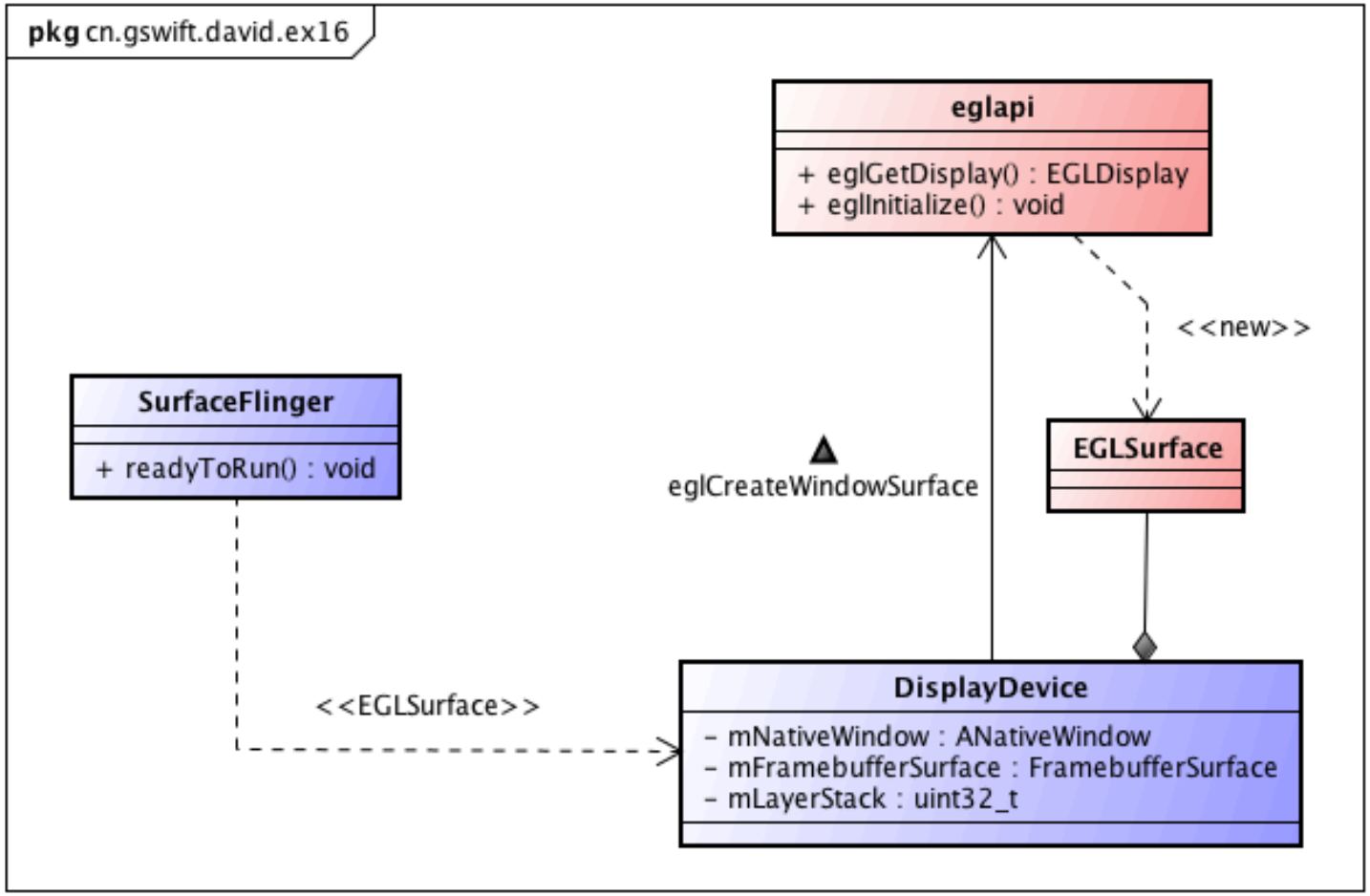
hgl : hardware OpenGL|ES , agl : android software OpenGL|ES render

Source from: http://0xlab.org/

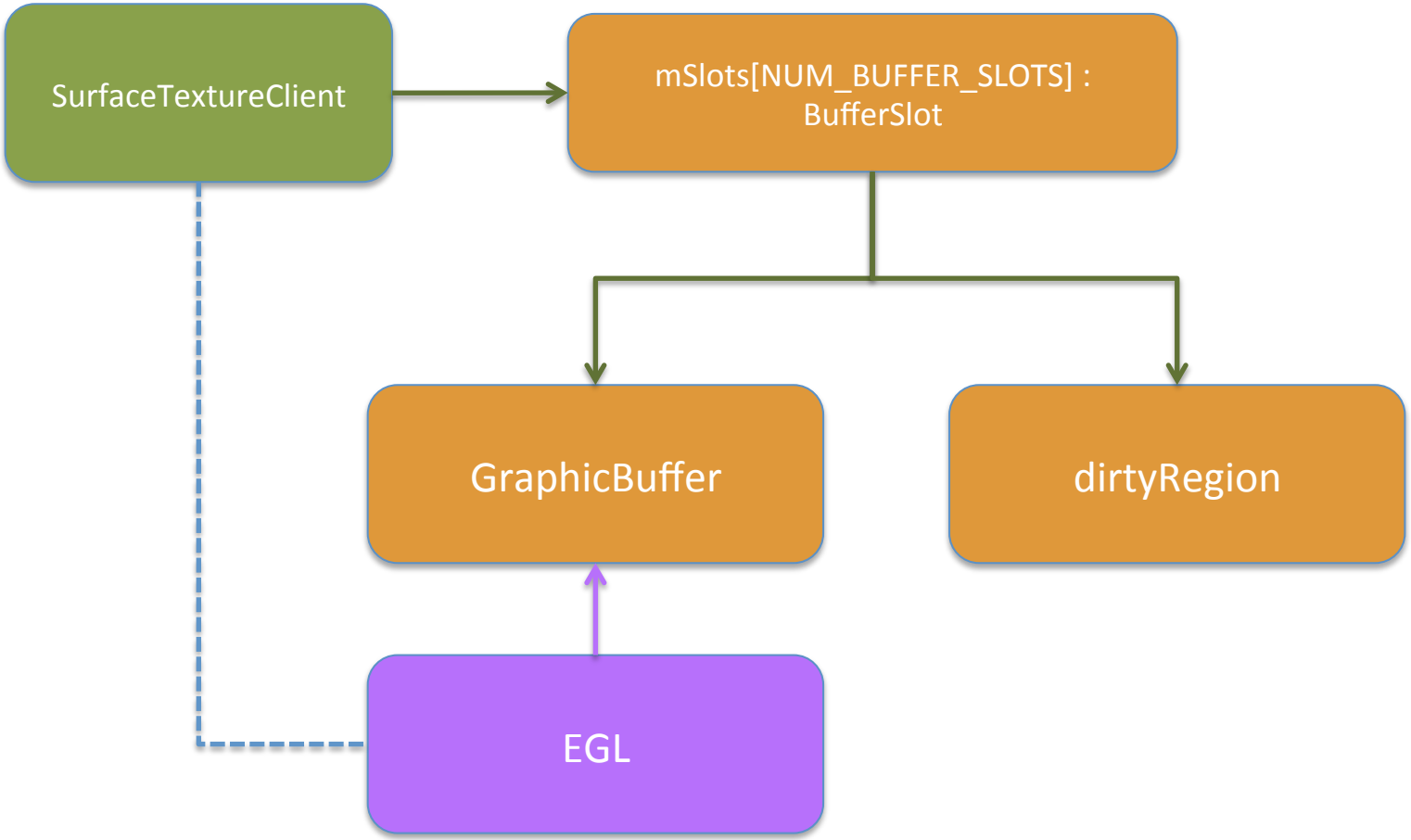
Call OpenGL API in SurfaceFlinger

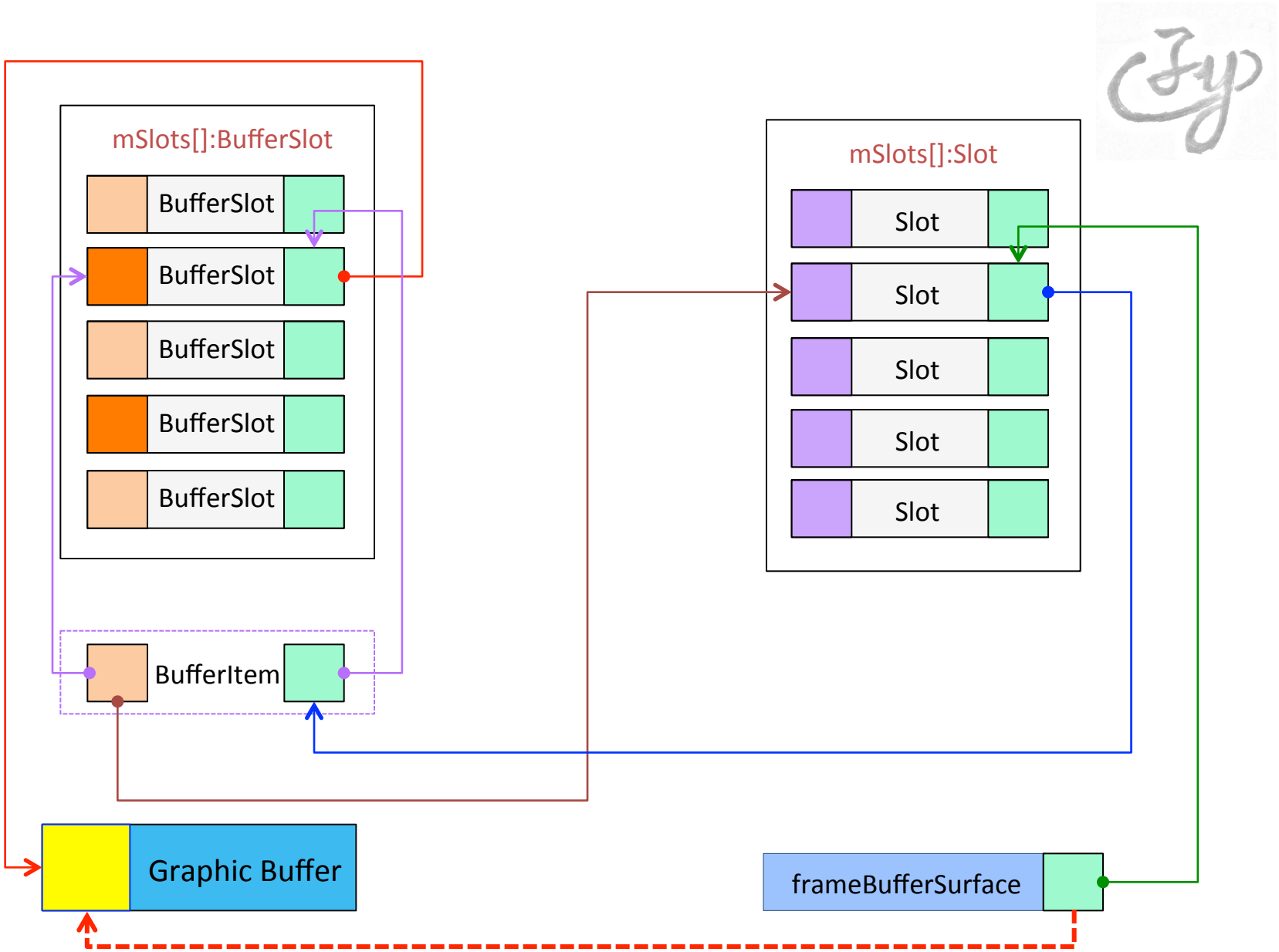


Create a EGLSurface

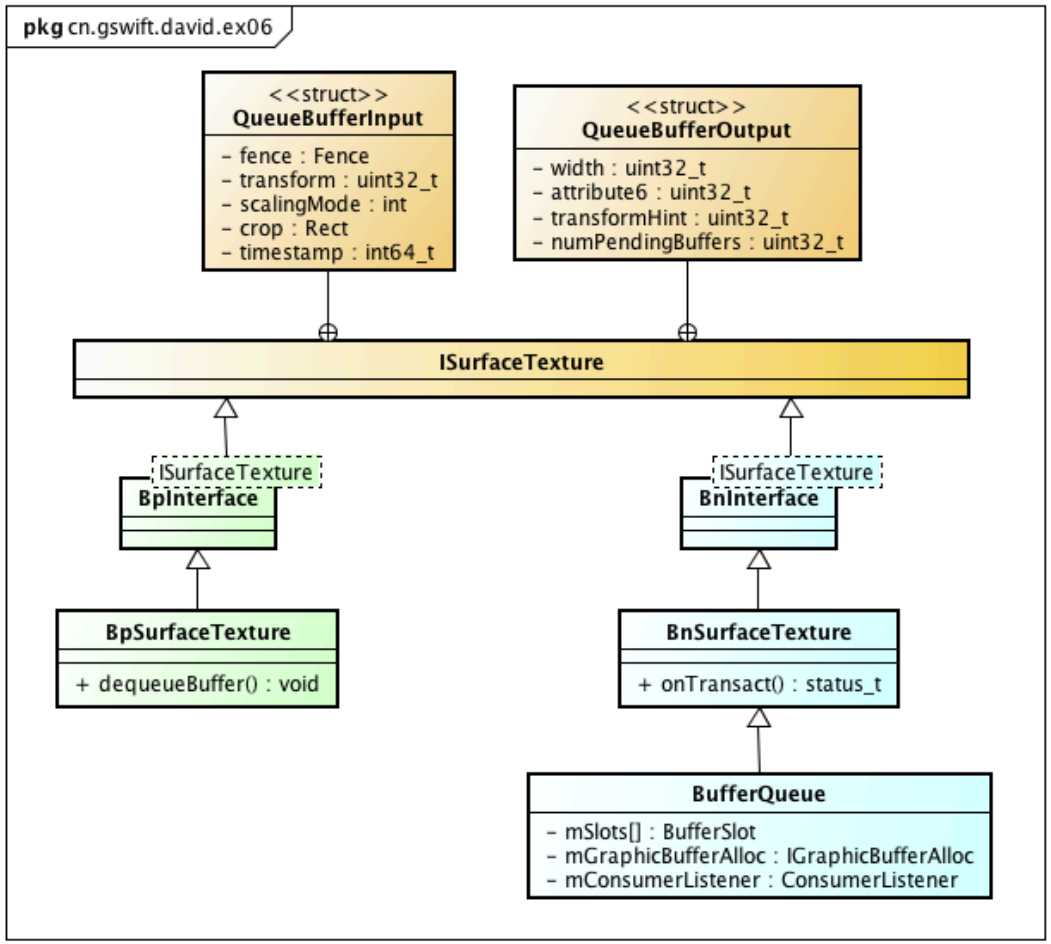


dequeueBuffer

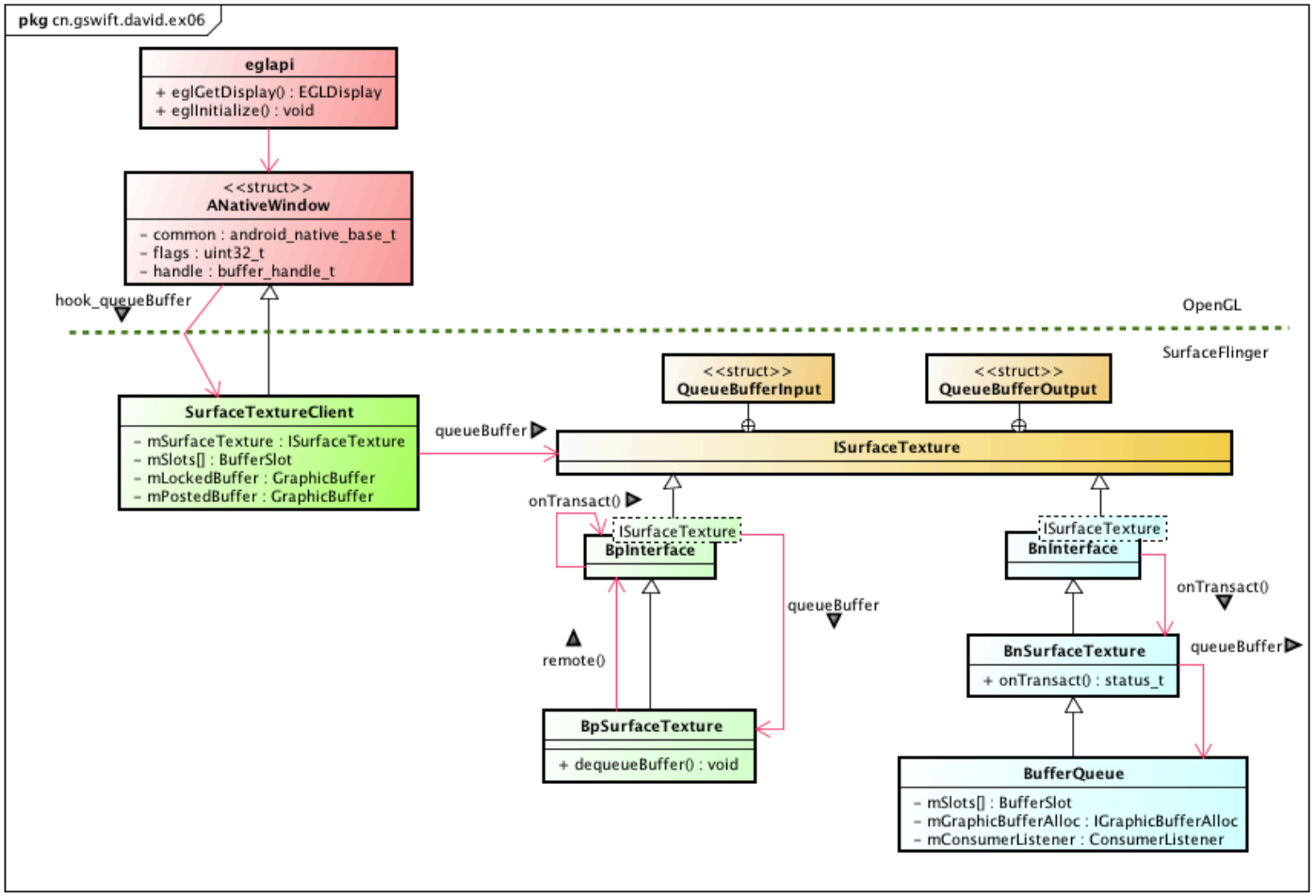



mSlots[]:BufferSlot
BufferSlot
BufferSlot
BufferSlot
BufferSlot
BufferSlot
BufferItem
mSlots[]:Slot
Slot
Slot
Slot
Slot
Slot
Graphic Buffer
frameBufferSurface

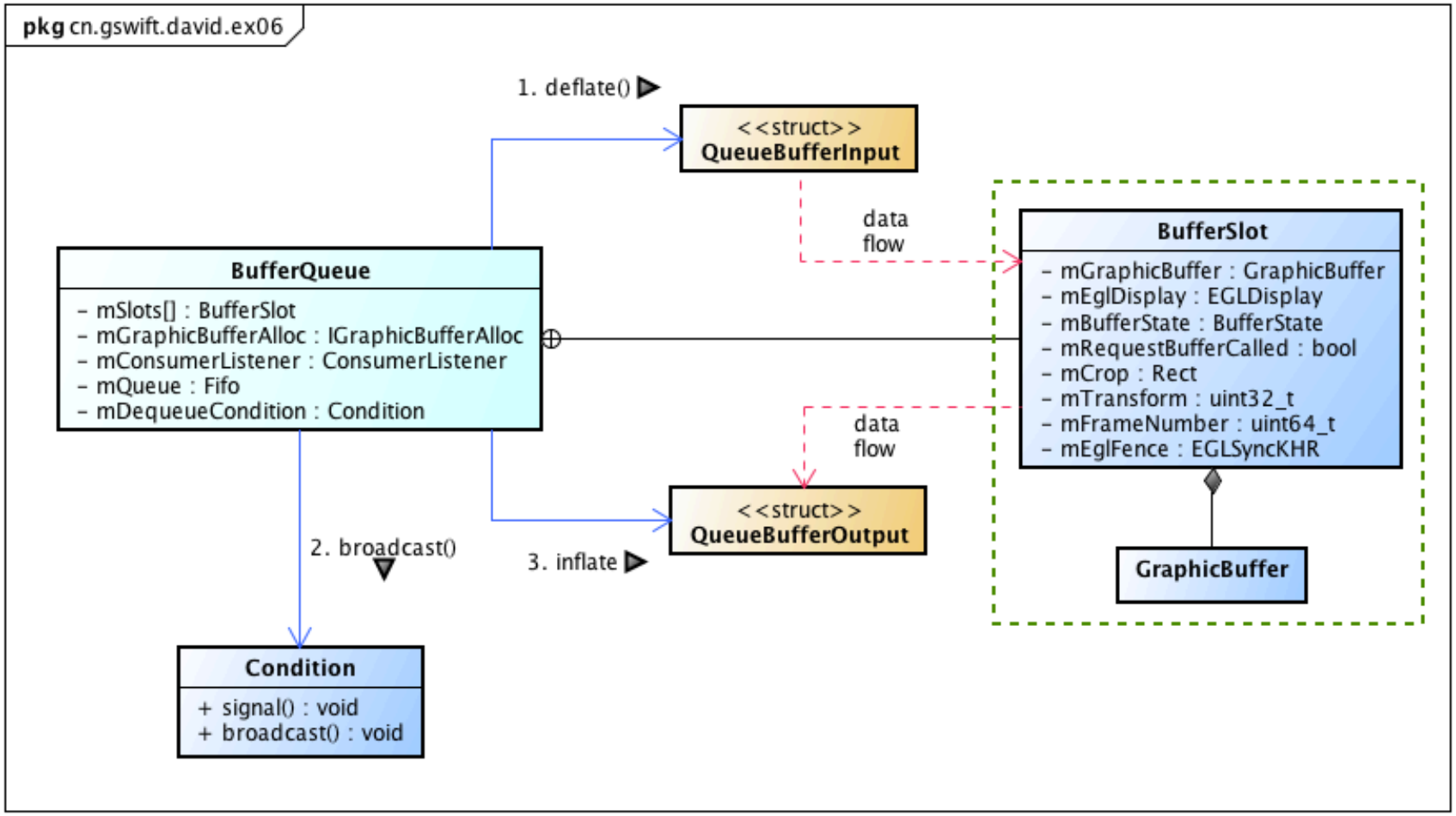
ISurfaceTexture



OpenGL hooks the queue buffer



BufferQueue queueBuffer()



```
status_t BufferQueue::queueBuffer(int buf,
        const QueueBufferInput& input, QueueBufferOutput* output) {
    ATRACE_CALL();
    ATRACE_BUFFER_INDEX(buf);

    Rect crop;
    uint32_t transform;
    int scalingMode;
    int64_t timestamp;
    sp<Fence> fence;

    input.deflate(&timestamp, &crop, &scalingMode, &transform, &fence);

        mSlots[buf].mTimestamp = timestamp;
        mSlots[buf].mCrop = crop;
        mSlots[buf].mTransform = transform;
        mSlots[buf].mFence = fence;

        switch (scalingMode) {
            case NATIVE_WINDOW_SCALING_MODE_FREEZE:
            case NATIVE_WINDOW_SCALING_MODE_SCALE_TO_WINDOW:
            case NATIVE_WINDOW_SCALING_MODE_SCALE_CROP:
                break;
            default:
                ST_LOGE("unknown scaling mode: %d (ignoring)", scalingMode);
                scalingMode = mSlots[buf].mScalingMode;
                break;
        }

        mSlots[buf].mBufferState = BufferSlot::QUEUED;
        mSlots[buf].mScalingMode = scalingMode;
        mFrameCounter++;
        mSlots[buf].mFrameNumber = mFrameCounter;

        mBufferHasBeenQueued = true;
        mDequeueCondition.broadcast();

        output->inflate(mDefaultWidth, mDefaultHeight, mTransformHint,
                mQueue.size());

        ATRACE_INT(mConsumerName.string(), mQueue.size());
    } // scope for the lock


    return OK;
}
```

deflate

inflate

参考资料：

Android4.2 http://code.metager.de/source/xref/android/4.2/

An Overview of Surfaceflinger, Rahul Bangalore, India, http://rahulonblog.blogspot.com/2013/06/an-overview-of-surfaceflinger.html

Android Graphic, Jim Huang , Sep 24, 2011 , http://0xlab.org/

GUI显示系统之SurfaceFlinger http://blog.csdn.net/uiop78uiop78

Android GUI 的更新过程，李先静，2010，http://www.limodev.cn/blog

Android 4.1 Surface系统变化说明 http://blog.csdn.net/innost

About Me

- I have been working as a product-designer specializing in software/Web application design and development. I am passionate about mobile application development and became interested in Android programming when the platform was launched by Google. Thus I was not programming on Android projects, I spent spare time reading technical blogs, researching, analyzing, and testing mobile applications, as a software consultancy specialized in android technologies.
- In my product-design time, in the developing, I've encountered too many program manage troubles that suffer due to poor communication and code design, I know that help them to understand the system framework is very important. I amd experienced in system and application layers, my goal is simple: help someone who wishes to better understand the **Android framework** in java、JNI and C/C++ libraries.

- Please also check my article and slides on this http://blog.sina.com.cn/gswift

Contact: Zhiyong.liu@aliyun.com

http://weibo.com/gswift