Debugging blink objects

pdr@chromium.org, wangxianzhu@chromium.org, masonf@chromium.org

Here is a collection of code snippets and flags for debugging blink objects (see also: dumping firefox data).

Geometry types

DOM nodes and the DOM tree

LayoutObjects and the Layout tree

Fragment tree

General paint debugging

PaintLayer tree

Blink property trees

Paint Chunks and display item list

Main thread cc::Layer contents (paint records)

Main thread cc::Layers and cc property trees

Compositor thread cc::LayerImpls, cc property trees, RenderPasses and quads

Printing a stacktrace

Debugging Skia pictures (skp)

How to edit this doc

Geometry types

Print geometry types such as LayoutRect, FloatRect, etc, in C++ with:

```
LayoutRect layout_rect(1, 2, 3, 4);
LOG(INFO) << "layout rect: " << layout_rect;
FloatPoint float_point(3, 1);
LOG(INFO) << "float point: " << float_point;
AffineTransform translation = AffineTransform::Translation(7, 9);
LOG(INFO) << "translation: " << translation;

// Output:
layout rect: "1,2 3x4"
float point: "3,1"
translation: "translation(7,9)"
```

In addition to << stream operators, most blink types also have ToString() functions:

```
std::vector<LayoutPoint> points({LayoutPoint(), LayoutPoint(1, 2), LayoutPoint(3, 4)});
StringBuilder builder;
for (const auto& point : points)
  builder.Append(String::Format("%s(%s)", builder.length() ? ", ": "", point.ToString().Utf8().data()));
LOG(INFO) << "points: " << builder.ToString();

// Output:
points: "(0,0), (1,2), (3,4)"
```

For code size reasons, gfx types (gfx::Rect, etc) do not have << stream operators and only have ToString():

```
gfx::Rect rect(1, 2, 3, 4);
LOG(INFO) << "rect: " << rect.ToString();

// Output:
rect: 1,2 3x4
```

DOM nodes and the DOM tree

To print a DOM node in C++, use:

```
LOG(INFO) << "root node: " << GetLayoutView()->GetNode();
if (Element* element = GetFrame().GetDocument()->getElementById("foo")) {
  LOG(INFO) << "foo element: " << element;
  // Can also use ShowNode(element);
}

// Output:
root node: #document
foo element: DIV id="foo" style="background: rebeccapurple;"
```

To print the DOM tree for a given object, use:

```
if (Element* element = GetFrame().GetDocument()->getElementById("foo"))
  ShowTree(element);

// Output:
BODY
*       DIV id="foo" style="background: rebeccapurple;"
                #text "\n  hello\n  "
                SPAN id="bar"
                        #text "world"
                #text "\n  "
                SPAN id="bold" style="font-weight: bold;"
                        #text "!"
```

```
              #text "\n"
       #text "\n"
```

It can also be useful to print the HTML for a given Element:

```
if (Element* element = GetFrame().GetDocument()->getElementById("foo"))
  LOG(INFO) << "foo outerHtml: " << element->outerHTML().Utf8().data();

// Output:
foo outerHtml:
<div id="foo" style="background: rebeccapurple;">
  hello
  <span id="bar">world</span>
  <span id="bold" style="font-weight: bold;">!</span>
</div>
```

LayoutObjects and the Layout tree

To print a LayoutObject in C++ use:

```
const LayoutObject* example = GetLayoutView();
LOG(INFO) << "example: " << example;

// Output:
example: 0x1944024270:LayoutBlockFlow      DIV id="example"
```

To print the LayoutObject tree for a given object:

```
GetLayoutView()->ShowLayoutTreeForThis();

// Output:
*LayoutView 0x4014204010                   #document
  LayoutBlockFlow 0x4014224010             HTML
    LayoutBlockFlow 0x4014224140           BODY
      LayoutBlockFlow 0x4014224270         DIV id="example" style="background: green;"
        LayoutText 0x4014234010            #text "\n  hi\n"
```

To print the entire LayoutObject tree before the prepaint lifecycle phase, run content shell with --vmodule=*pre_paint_tree_walk*=3.
Note: this requires a build with "dcheck_always_on = true". When using chrome instead of content shell, --enable-logging=stderr is needed. On windows "--enable-logging --v=0 --no-sandbox" is needed.

Fragment tree

To print a fragment tree in C++ use:

```
NGPhysicalFragment::ShowFragmentTree(*GetLayoutView());

// Output:
Box (block-flow-root block-flow)(self paint) offset:unplaced size:800x34 LayoutNGBlockFlow HTML
  Box (block-flow) offset:8,8 size:784x18 LayoutNGBlockFlow BODY
    Box (block-flow children-inline) offset:0,0 size:784x18 LayoutNGBlockFlow DIV id='example'
      NGPhysicalLineBoxFragment offset:0,0 size:12.4531x18
        NGPhysicalTextFragment 'hi' offset:0,0 size:12.4531x17
```

To print the entire fragment tree before the prepaint lifecycle phase, run content shell with --vmodule=*pre_paint_tree_walk*=3.
Note: this requires a build with "dcheck_always_on = true". When using chrome instead of content shell, --enable-logging=stderr is needed. On windows "--enable-logging --v=0 --no-sandbox" is needed.

General paint debugging

Run content shell with: --vmodule=*paint*=2
This will print the PaintLayer tree, property trees, and display item lists.

For more verbose output, run with: --vmodule=*paint*=3
This will print the LayoutObject tree, the PaintLayer tree, property trees, and display item lists.

Windows may require "--enable-logging" and/or "--no-sandbox" for --vmodule to work, see: https://www.chromium.org/for-testers/enable-logging.

PaintLayer tree

Run content shell with: --vmodule=*pre_paint_tree_walk*=2 if CompositeAfterPaint is not enabled, or --vmodule=*cull_rect_updater*=2 otherwise.
Note: this requires a build with "dcheck_always_on = true". When using chrome instead of content shell, --enable-logging=stderr is needed. On windows "--enable-logging --v=0 --no-sandbox" is needed.

Or, in C++, at the end of LocalFrameView::PaintTree, add:

```
PaintLayer* root_layer = layout_view->Layer();
while (root_layer->Parent())
  root_layer = root_layer->Parent();
showLayerTree(root_layer);
```

```
// Output:
*layer 0x3d95814010 at (0,0) size 800x600 (composited, bounds=at (0,0) size 800x600, drawsContent=0)
  LayoutView 0x3d95804010 at (0,0) size 800x600
     paint_offset=(0,0) visual_rect=(0,0 800x600) state=(t:0x147c5bc610 c:0x147c5ac510 e:0x147c5ac290)
 positive z-order list(1)
```

```
   layer 0x3d958140e8 at (0,0) size 800x34
    LayoutBlockFlow 0x3d95824010 {HTML} at (0,0) size 800x34
       paint_offset=(0,0) visual_rect=(0,0 800x34) state=(t:0x147c5bc790 c:0x147c5ac8d0 e:0x147c5ac290)
      LayoutBlockFlow 0x3d95824140 {BODY} at (8,8) size 784x18
         paint_offset=(8,8) visual_rect=(8,8 784x18)
   positive z-order list(1)
     layer 0x3d958141c0 at (8,8) size 784x18 (composited, bounds=at (0,0) size 784x18, drawsContent=1)
      LayoutBlockFlow 0x3d95824270 {DIV} at (0,0) size 784x18 [bgcolor=#008000] id="example"
         paint_offset=(0,0) visual_rect=(0,0 784x18) state=(t:0x147c5bca90 c:0x147c5ac8d0 e:0x147c5ac290)
        LayoutText 0x3d95834010 {#text} at (0,0) size 13x17
           paint_offset=(0,0) visual_rect=(0,-1 13x19)
          text run at (0,0) width 13: "hi"
```

Blink property trees

Run content shell with: --vmodule=*paint*=1
Note: this requires a build with "dcheck_always_on = true". When using chrome instead of content shell, --enable-logging=stderr is needed. On windows "--enable-logging --v=0 --no-sandbox" is needed.

Or, in C++:

```
#include "third_party/blink/renderer/core/paint/paint_property_tree_printer.h"
...
bool LocalFrameView::RunPrePaintLifecyclePhase(
    DocumentLifecycle::LifecycleState target_state) {
  ...
  showAllPropertyTrees(*this);
  return target_state > DocumentLifecycle::kPrePaintClean;
}
```

```
// Output:
Transform tree:
root 0x4682bbc010 {"flattensInheritedTransform":false,"scroll":"0x4682bac3d0"}
   0x4682bbc190 {"parent":"0x4682bbc010","flattensInheritedTransform":false,"compositorElementId":"(27)"}
    VisualViewport Scale Node 0x4682bbc310 {"parent":"0x4682bbc190","compositorElementId":"(16)"}
      VisualViewport Translate Node 0x4682bbc490 {"parent":"0x4682bbc310",
          "flattensInheritedTransform":false,"scroll":"0x4682bac650"}
        PaintOffsetTranslation (LayoutView #document) 0x4682bbc610 {"parent":"0x4682bbc490","changed":true}
          ScrollTranslation (LayoutView #document) 0x4682bbc790 {"parent":"0x4682bbc610","changed":true,
              "directCompositingReasons":"rootScroller","scroll":"0x4682bac790"}
            PaintOffsetTranslation (LayoutBlockFlow DIV id='example') 0x4682bbc910 {
                "parent":"0x4682bbc790","changed":true,"matrix":"translation(8,8,0)"}
              Transform (LayoutBlockFlow DIV id='example') 0x4682bbca90 {"parent":"0x4682bbc910"}

Clip tree:
root 0x4682bac510 {"localTransformSpace":"0x4682bbc010","rect":"InfiniteIntRect"}
  OverflowClip (LayoutView #document) 0x4682bac8d0 {"parent":"0x4682bac510","changed":true,
      "localTransformSpace":"0x4682bbc610","rect":"0,0 800x600",
      "rectExcludingOverlayScrollbars":"0,0 800x600"}

Effect tree:

Scroll tree:
root 0x4682bac3d0 {}
  VisualViewport Scroll Node 0x4682bac650 {"parent":"0x4682bac3d0","containerRect":"0,0 800x600",
      "contentsSize":"800x600","userScrollable":"both","scrollsInnerViewport":"true",
      "maxScrollOffsetAffectedByPageScale":"true","compositorElementId":"(18)"}
    Scroll (LayoutView #document) 0x4682bac790 {"parent":"0x4682bac650","containerRect":"0,0 800x600",
        "contentsSize":"800x600","userScrollable":"both"}
```

Paint Chunks and display item list

Run content shell with: --vmodule=*paint_controller*=1
Note: this requires a build with "dcheck_always_on = true". When using chrome instead of content shell, --enable-logging=stderr is needed. On windows "--enable-logging --v=0 --no-sandbox" is needed.

```
// Output:
current display item list: [
  {
    "chunk": "LayoutBlockFlow DIV id='example'
        0x1b94c141c0:LayoutBlockFlow DIV id='example':DrawingPaintPhaseSelfBlockBackgroundOnly:0",
    "state": "t:0x3e697bca90 c:0x3e697ac8d0 e:0x3e697ac290",
    "displayItems": [
      "0x1b94c24278:LayoutBlockFlow DIV id='example':DrawingBoxDecorationBackground:0",
      "0x1b94c44010:InlineTextBox 'hi':DrawingPaintPhaseForeground:0"
    ]
  }
]
new display item list: [ ... ]
```

To print more information such as the visual rects, use --vmodule=*paint_controller*=2:

```
// Output:
current display item list: [
  {
    "chunk": "LayoutBlockFlow DIV id='example'
        0x1b94c141c0:LayoutBlockFlow DIV id='example':DrawingPaintPhaseSelfBlockBackgroundOnly:0",
    "state": "t:0x3e697bca90 c:0x3e697ac8d0 e:0x3e697ac290",
    "displayItems": [
      {
        "index": 0,
        "clientDebugName": "LayoutBlockFlow DIV id='example'",
        "id": "0x1b94c24278:LayoutBlockFlow DIV id='example':DrawingBoxDecorationBackground:0",
        "visualRect": "0,0 784x18",
        "opaque": false
      },
      {
        "index": 1,
        "clientDebugName": "InlineTextBox 'hi'",
        "id": "0x1b94c44010:InlineTextBox 'hi':DrawingPaintPhaseForeground:0",
        "visualRect": "0,-1 13x19",
        "opaque": false
      }
    ]
  }
]
```

To print even more information such as the paint records, use --vmodule=*paint_controller*=3:

```
// Output:
current display item list: [
  {
    "chunk": "LayoutBlockFlow DIV id='example'
        0x1b94c141c0:LayoutBlockFlow DIV id='example':DrawingPaintPhaseSelfBlockBackgroundOnly:0",
    "state": "t:0x3e697bca90 c:0x3e697ac8d0 e:0x3e697ac290",
    "displayItems": [
      {
        "index": 0,
        "clientDebugName": "InlineTextBox 'hi'",
        "id": "0x1605c60410:InlineTextBox 'hi':DrawingPaintPhaseForeground:0",
        "visualRect": "401,357 72x20",
        "opaque": false,
        "record": [
          {
            "method": "drawTextBlob",
            "params": {
              "x": 401.5,
              "y": 373,
              "paint": {
                "color": "#FF333333",
                "strokeWidth": 0,
                "strokeMiter": 4,
                "flags": "AntiAlias",
                "filterLevel": "Low",
                "strokeCap": "Butt",
                "strokeJoin": "Miter",
                "styleName": "Fill"
              }
            }
          }
        ]
      }
    ]
  }
]
```

To further debug the paint record (a Skia picture), see: Debugging Skia Pictures (skp).

Main thread cc::Layer contents (paint records)

To print the contents (paint ops) in each cc::Layer after compositing decisions have been made in PaintArtifactCompositor, run content shell with: --vmodule=paint_artifact_compositor=3
Note: this requires a build with "dcheck_always_on = true". When using chrome instead of content shell, --enable-logging=stderr is needed. On windows "--enable-logging --v=0 --no-sandbox" is needed.

```
Composited layers:
{
  "layers": [
    {
      "ccLayerId": 6,
      "name": "LayoutView #document",
      "bounds": [1600, 1200],
      "drawsContent": false,
      "compositingReasons": [
        "Is a scrollable overflow element"
      ]
    },
    {
      "ccLayerId": 7,
      "name": "Scrolling background of LayoutView #document",
      "bounds": [1600, 1200],
      "contentsOpaque": true,
      "backgroundColor": "#FFFFFF",
      "compositingReasons": [
        "Is a scrollable overflow element",
        "Is the document.rootScroller"
      ],
      "paintChunkContents": [
        {
          "data": "PaintChunk(begin=0, end=43, id=... cacheable=1 props=(...) bounds=0,0 1600x1200 ...)",
          "displayItems": [
            "0: Scrolling background of LayoutView #document:DrawingDocumentBackground:0",
            "1: LayoutNGBlockFlow DIV class='big':DrawingBoxDecorationBackground:0",
            "2: LayoutNGBlockFlow DIV class='big':DrawingBoxDecorationBackground:0",
            ...
          ]
        }
      ],
      "paintRecord": [
        {
          "method": "drawRect",
          "params": {
            "rect": {
              "left": 0,
              "top": 0,
              "right": 1600,
              "bottom": 1200
            },
            "paint": {
              "color": "#FFFFFFFF",
              "strokeWidth": 0,
              "strokeMiter": 4,
              "flags": "AntiAlias",
              "strokeCap": "Butt",
              "strokeJoin": "Miter",
              "styleName": "Fill",
              "blendMode": "Src"
            }
          }
        },
        ...
```
To further debug the paint record (a Skia picture), see: Debugging Skia Pictures (skp).

Main thread cc::Layers and cc property trees

To print blink cc::Layers and cc property trees, run content shell with: --vmodule=layer_tree_view=3
Note: this requires a build with "dcheck_always_on = true". When using chrome instead of content shell, --enable-logging=stderr is needed. On windows "--enable-logging --v=0 --no-sandbox" is needed.

(prints from LayerTreeView::DidUpdateLayers)

```
// Output:
After updating layers:
property trees:
{
   "clip_tree": {
      "nodes": [ {
         "clip": [ 0, 0, 0, 0 ],
         "clip_type": 0,
         "id": 0,
         "parent_id": -1,
         "transform_id": -1
      }, ...
   },
   "effect_tree": {
       ...
   },
   "scroll_tree": {
      "nodes": [ {
         "bounds": {
            "height": 0,
            "width": 0
         },
         "container_bounds": {
            "height": 0,
            "width": 0
         },
         "element_id": {
            "id_": 0
         },
         "id": 0,
         "offset_to_transform_parent": [ 0, 0 ],
         "overscroll_behavior_x": 1,
         "overscroll_behavior_y": 1,
         "parent_id": -1,
         "scrollable": false,
         "should_flatten": false,
         "transform_id": 0,
         "user_scrollable_horizontal": false,
         "user_scrollable_vertical": false
      }, ...
   },
   "sequence_number": 1,
   "transform_tree": {
      "nodes": [ {
         "element_id": {
            "id_": 0
         },
         "flattens_inherited_transform": 0,
         "id": 0,
         "local": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
          "parent_id": -1,
          "post_local": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
          "pre_local": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
          "scroll_offset": [ 0, 0 ],
          "snap_amount": [ 0, 0 ],
          "sorting_context_id": 0,
          "source_node_id": -1
       }, ...
    }
}

cc:Layers:
layer_id: 1
  name: Root Transform Layer
  Bounds: 0x0
  ElementId: (0)
  OffsetToTransformParent: [0.000000 0.000000]
  Position: 0.000000,0.000000
  scrollable: 0
  clip_tree_index: 1
  effect_tree_index: 1
  scroll_tree_index: 1
  transform_tree_index: 1

layer_id: 2
  name: LayoutView #document
  Bounds: 800x600
  ElementId: (0)
  OffsetToTransformParent: [0.000000 0.000000]
  Position: 0.000000,0.000000
  scrollable: 0
  clip_tree_index: 2
  effect_tree_index: 2
  scroll_tree_index: 1
  transform_tree_index: 1
  ...
```

It is also possible to print a cc::Layer in C++ using cc::Layer::ToString():

```
bool LayerTreeHost::DoUpdateLayers(Layer* root_layer) {
  ...
  UpdateHudLayer(debug_state_.ShowHudInfo());
  if (hud_layer())
    LOG(INFO) << "LayerTreeHost::hud_layer(): " << hud_layer()->ToString();
  ...

// Output:
LayerTreeHost::hud_layer(): layer_id: 21
  Bounds: 256x256
  ElementId: (0)
  OffsetToTransformParent: [0.000000 0.000000]
  Position: 0.000000,0.000000
  scrollable: 0
  clip_tree_index: 1
  effect_tree_index: 1
  scroll_tree_index: 1
  transform_tree_index: 1
```

To print main-thread cc::Layers and cc property trees for the ui compositor (e.g., tabs, etc), use:
    --vmodule=*ui/compositor*=3
Similarly, for the android compositor:
    --vmodule=compositor_impl_android=3
Note: these require a build with "dcheck_always_on = true" and on windows "--enable-logging --v=0 --no-sandbox" is needed.

Compositor thread cc::LayerImpls, cc property trees, RenderPasses and quads

Run content shell with:
  --vmodule=layer_tree_host_impl=3 to log from the renderer processes only
or
  –vmodule=layer_tree_host_impl=4 to log from all processes.
Note: When using chrome instead of content shell, --enable-logging=stderr is needed. On windows "--enable-logging --v=0 --no-sandbox" is needed.

cc::LayerImpls and property trees after pushing main->pending, printed from LayerTreeHostImpl::FinishCommit

```
// Output:
Renderer: After finishing commit on impl, the sync tree:
property_trees:
{
   "clip_tree": {
      "nodes": [ {
         "clip": [ 0, 0, 0, 0 ],
         "clip_type": 0,
         "id": 0,
         "parent_id": -1,
         "transform_id": -1
      }, ...
   },
   "effect_tree": {
       ...
   },
   "scroll_tree": {
      "nodes": [ {
         "bounds": {
            "height": 0,
            "width": 0
         },
         "container_bounds": {
            "height": 0,
            "width": 0
         },
         "element_id": {
            "id_": 0
         },
         "id": 0,
         "offset_to_transform_parent": [ 0, 0 ],
         "overscroll_behavior_x": 1,
         "overscroll_behavior_y": 1,
         "parent_id": -1,
         "scrollable": false,
         "should_flatten": false,
         "transform_id": 0,
         "user_scrollable_horizontal": false,
         "user_scrollable_vertical": false
      }, ...
   },
   "sequence_number": 1,
   "transform_tree": {
      "nodes": [ {
         "element_id": {
            "id_": 0
         },
         "flattens_inherited_transform": 0,
         "id": 0,
         "local": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
         "parent_id": -1,
         "post_local": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
         "pre_local": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
         "scroll_offset": [ 0, 0 ],
         "snap_amount": [ 0, 0 ],
         "sorting_context_id": 0,
         "source_node_id": -1
      }, ...
   }
}

cc::LayerImpls:
[ {
   "Bounds": [ 0, 0 ],
   "ContentsOpaque": false,
   "DrawsContent": false,
   "HitTestableWithoutDrawsContent": false,
   "Is3dSorted": false,
   "LayerId": 1,
   "LayerType": "cc::PictureLayerImpl",
   "OffsetToTransformParent": [ 0, 0 ],
   "Opacity": 1,
   "Transform": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
   "clip_tree_index": 1,
   "effect_tree_index": 1,
   "scroll_tree_index": 1,
   "transform_tree_index": 1
}, {
   "Bounds": [ 800, 600 ],
   "ContentsOpaque": false,
   "DrawsContent": false,
   "HitTestableWithoutDrawsContent": false,
   "Is3dSorted": false,
   "LayerId": 2,
   "LayerType": "cc::PictureLayerImpl",
   "OffsetToTransformParent": [ 0, 0 ],
   "Opacity": 1,
   "Transform": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
   "clip_tree_index": 2,
   "effect_tree_index": 2,
   "scroll_tree_index": 1,
   "transform_tree_index": 1
}, ...
```

cc::LayerImpls and property trees after pushing pending->active, printed from LayerTreeHostImpl::ActivateSyncTree

```
// Output:
Renderer: After activating sync tree, the active tree:
property_trees:
{
   "clip_tree": {
      ...
   },
   "effect_tree": {
      ...
   },
   "scroll_tree": {
      ...
   },
   "transform_tree": {
      ...
   }
}

cc::LayerImpls:
[ {
   "Bounds": [ 0, 0 ],
   "ContentsOpaque": false,
   "DrawsContent": false,
   "HitTestableWithoutDrawsContent": false,
   "Is3dSorted": false,
   "LayerId": 1,
   "LayerType": "cc::PictureLayerImpl",
   "OffsetToTransformParent": [ 0, 0 ],
   "Opacity": 1,
   "Transform": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
   "clip_tree_index": 1,
   "effect_tree_index": 1,
   "scroll_tree_index": 1,
   "transform_tree_index": 1
}, {
   "Bounds": [ 800, 600 ],
   "ContentsOpaque": false,
   "DrawsContent": false,
   "HitTestableWithoutDrawsContent": false,
   "Is3dSorted": false,
   "LayerId": 2,
   "LayerType": "cc::PictureLayerImpl",
   "OffsetToTransformParent": [ 0, 0 ],
   "Opacity": 1,
   "Transform": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
   "clip_tree_index": 2,
   "effect_tree_index": 2,
   "scroll_tree_index": 1,
   "transform_tree_index": 1
}, ...
```

RenderPasses and quads, printed from LayerTreeHostImpl::PrepareToDraw

```
Renderer: Prepare to draw
{
   "has_no_damage": false,
   "render_passes": [ {
      "backdrop_filters": [  ],
      "cache_render_pass": false,
      "cat": "disabled-by-default-viz.quads",
      "copy_requests": 0,
      "damage_rect": [ 0, 0, 800, 600 ],
      "filters": [  ],
      "generate_mipmap": false,
      "has_damage_from_contributing_content": true,
      "has_transparent_background": false,
      "id": "CompositorRenderPass/0xa1",
      "output_rect": [ 0, 0, 800, 600 ],
      "quad_list": [ {
          "background_color": "rgba(0.000000, 0.000000, 0.000000, 0.000000",
          "content_space_rect": [ 0, 15, 15, 34 ],
          "content_space_visible_rect": [ 0, 15, 15, 34 ],
          "is_video_frame": false,
          "material": 9,
          "nearest_neighbor": false,
          "needs_blending": true,
          "premultiplied_alpha": true,
          "protected_video_type": 0,
          "rect_as_target_space_quad": [ 785, 15, 800, 15, 800, 49, 785, 49 ],
          "rect_is_clipped": false,
          "resource_id": 2,
          "shared_state": {
             "id_ref": "0x35e800b63700"
          },
          "should_draw_with_blending": true,
          "uv_bottom_right": [ 1, 1 ],
          "uv_top_left": [ 0, 0 ],
          "vertex_opacity": [ 1, 1, 1, 1 ],
          "visible_rect_as_target_space_quad": [ 785, 15, 800, 15, 800, 49, 785, 49 ],
          "visible_rect_is_clipped": false,
          "y_flipped": false
       },
       ... ],
       "shared_quad_state_list": [ {
          "are_contents_opaque": false,
          "blend_mode": "SrcOver",
          "cat": "disabled-by-default-viz.quads",
          "de_jelly_delta_y": 0,
          "id": "viz::SharedQuadState/0x35e800b63700",
          "is_fast_rounded_corner": false,
          "layer_content_rect": [ 0, 0, 15, 600 ],
          "layer_visible_content_rect": [ 0, 0, 15, 600 ],
          "mask_filter_bounds": [ 0, 0, 0, 0 ],
          "opacity": 1,
          "sorting_context_id": 0,
          "transform": [ 1, 0, 0, 785, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ]
       },
       ... ],
       "subtree_capture_id": "SubtreeCaptureId(0)",
       "subtree_size": {
          "height": 0,
          "width": 0
       }
    } ]
 }
```

RenderPasses, printed from LayerTreeHostImpl::DrawLayers

```
Renderer: Submitting a frame:
render pass ids in order:
 161
rooted render pass tree:
(0x35e8003c66c0) render pass id=161 output_rect=0,0 800x600
  (0x35e800b63700) switched to sqs with opacity=1, blend_mode=kSrcOver quad_layer_rect=0,0 15x600
    DrawQuad, material: kTextureContent
    DrawQuad, material: kTextureContent
  (0x35e800b63828) switched to sqs with opacity=1, blend_mode=kSrcOver quad_layer_rect=0,0 785x10018
    (0x35e800c38230) SolidColorDrawQuad color=rgba(255, 255, 255, 255)
  (0x35e800c20000) switched to sqs with opacity=1, blend_mode=kSrcOver quad_layer_rect=0,0 800x600
    (0x35e800c38348) SolidColorDrawQuad color=rgba(255, 255, 255, 255)
```

It is also possible to print a cc::LayerImpl from C++ using cc::LayerImpl::ToString():

```
void LayerTreeImpl::PushPropertiesTo(LayerTreeImpl* target_tree) {
  if (hud_layer())
    LOG(INFO) << "LayerTreeImpl::hud_layer(): " << hud_layer()->ToString();
  ...

// Output:
LayerTreeImpl::hud_layer(): {
   "Bounds": [ 256, 256 ],
   "ContentsOpaque": false,
   "DrawsContent": true,
   "HitTestableWithoutDrawsContent": false,
   "Is3dSorted": false,
   "LayerId": 20,
   "LayerType": "cc::HeadsUpDisplayLayerImpl",
   "OffsetToTransformParent": [ 0, 0 ],
   "Opacity": 1,
   "Transform": [ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 ],
   "WheelRegion": [ 0, 0, 256, 256 ],
   "clip_tree_index": 1,
   "effect_tree_index": 1,
   "scroll_tree_index": 1,
   "transform_tree_index": 1
}
```

Printing a stacktrace

```
#include <base/debug/stack_trace.h>

// The argument is the number of stackframes to print. If no number is passed, all frames are printed.
base::debug::StackTrace(14).Print();

// Output:
0  base::debug::StackTrace::StackTrace(unsigned long) + 83
```

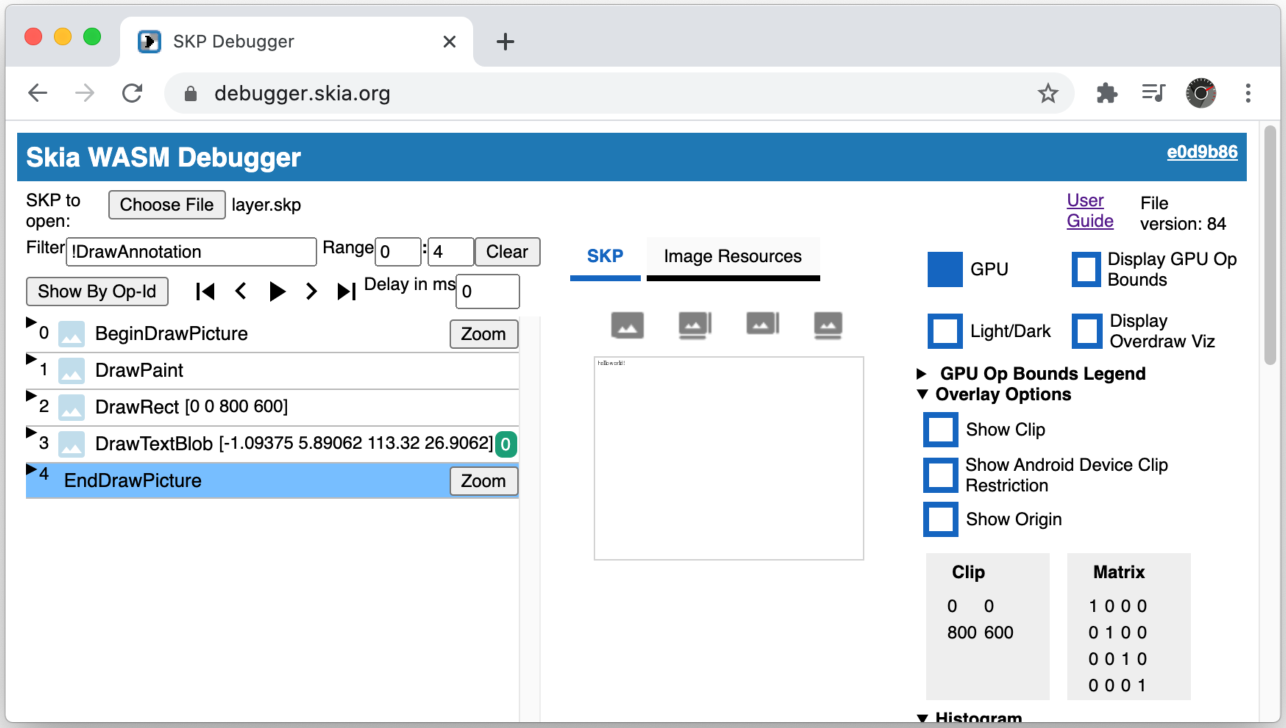
```
base::debug::StackTrace::StackTrace(unsigned long) + 29
base::debug::StackTrace::StackTrace() + 26
blink::LocalFrameView::RunPrePaintLifecyclePhase(blink::DocumentLifecycle::LifecycleState) + 64
blink::LocalFrameView::UpdateLifecyclePhasesInternal(blink::DocumentLifecycle::LifecycleState) + 835
blink::LocalFrameView::UpdateLifecyclePhases(blink::DocumentLifecycle::LifecycleState, ...) + 1187
blink::LocalFrameView::UpdateAllLifecyclePhases(blink::DocumentLifecycle::LifecycleUpdateReason) + 56
blink::PageAnimator::UpdateAllLifecyclePhases(blink::LocalFrame&, ...) + 91
blink::PageWidgetDelegate::UpdateLifecycle(blink::Page&, blink::LocalFrame&, ...) + 118
blink::WebViewImpl::UpdateLifecycle(blink::WebWidget::LifecycleUpdate, ...) + 338
blink::WebViewFrameWidget::UpdateLifecycle(blink::WebWidget::LifecycleUpdate, ...) + 45
content::RenderWidget::UpdateVisualState(bool) + 101
content::LayerTreeView::UpdateLayerTreeHost(bool) + 42
cc::LayerTreeHost::RequestMainFrameUpdate(bool) + 42
```

Debugging Skia pictures (skp)

Follow these instructions to save the Skia pictures (skp files) from a page:

1. Run content shell with "--enable-gpu-benchmarking --no-sandbox"
2. Open devtools (right-click > inspect) and select the console tab
3. Type the following and press enter:
   chrome.gpuBenchmarking.printToSkPicture('/tmp/skiatest')

Then open the Skia Debugger (https://debugger.skia.org) and open the saved .skp file in /tmp/skiatest/



SKP to open: layer.skp

File version: 84

Filter: !DrawAnnotation   Range: 0 : 4   Delay in ms: 0

0 BeginDrawPicture
1 DrawPaint
2 DrawRect [0 0 800 600]
3 DrawTextBlob [-1.09375 5.89062 113.32 26.9062] 0
4 EndDrawPicture

Clip	Matrix
0 0	1 0 0 0
800 600	0 1 0 0
	0 0 1 0
	0 0 0 1

How to edit this doc

The code snippets in this doc were made using the Google Docs code blocks add-on with language "cpp" and theme "default". To use this, go to Extensions > Add Ons > View document addons, and use the "Code Blocks" add on.
This is not editable by all to prevent spam, just request edit permissions from pdr@chromium.org if you'd like to edit.