# Removing the Blink web/ layer

slangley@chromium.org
7-April-2017
Status: Publishd
http://bit.ly/2njib8d

## Objective

Remove most* the Blink web/ layer by merging it into modules/, core/ & platform/ as necessary. The web/ layer introduces many now obsolete abstractions and dependency inversions that makes development on the blink codebase slow and cumbersome.

This work is a realization of principles and ideas outlined in (1), (2) & (3) below.

Note*: We may need to keep the web/ link unit for the startup files like WebKit.cpp, which glues together the various initialization units. The number of files that remain should be minimal.

## Background Reading

1. Blink Onion Soup
2. Blink Componentization
3. Proposal for a future Blink architecture

## Implementation Plan

### Reorganization of files and dependencies.

Ideally we can come up with a solution so that we are able to keep modules/ and core/ in separate link units. This is the simplest way to enforce the dependency pattern where code in modules/ can depend on core/, but code in core/ cannot depend on code in modules/.

To achieve this, there are generally 4 locations where files that currently reside in Source/web/ can be moved to:

1. **Source/core/(dom|frame|layout|*)** - Files that are implementations of abstract classes already defined in core/*, and only exist in web/ as they needed to interact with other web/ classes. For example EditorClientImpl.cpp,

2. **Source/modules/*** - Files that are implementations of abstract classes defined in modules/*, and only exist in web/ as they needed to interact with other web/ classes. For example IndexedDBClientImpl.cpp.
3. **Source/core/exported/** - Files that are implementations of abstract classes defined in Source/public/web/ that have no dependencies on modules/. For example WebSharedWorkerImpl.
4. **Source/modules/exported/** - Files that are implementations of abstract classes defined in Source/public/web/ that have dependencies on modules/. For example WebMediaDeviceChangeObserver.cpp

Naturally relevant test cases will follow the implementation that they test.

Now we are left with the cases where we would like to push implementation to core/web (so we can remove inverted code dependencies between core/ and either modules/ or web/) but the code has a dependency on modules/ - for example LocalFrameClientImpl.cpp.

In many cases it should be possible to introduce an observer pattern, where modules can register callbacks at initialization time that are invoked in place of the direct static calls that occur now.

For example, this code in LocalFrameClientImpl.cpp introduces dependencies on modules/.

```
if (document) {
   DeviceMotionController::from(*document);
   DeviceOrientationController::from(*document);
   DeviceOrientationAbsoluteController::from(*document);
   if (RuntimeEnabledFeatures::deviceLightEnabled())
     DeviceLightController::from(*document);
   NavigatorGamepad::from(*document);
   NavigatorServiceWorker::from(*document);
   DOMWindowStorageController::from(*document);
   if (RuntimeEnabledFeatures::webVREnabled() ||
       OriginTrials::webVREnabled(document->getExecutionContext()))
     NavigatorVR::from(*document);
}
```

We can replace this with callbacks like such:

1. Define the callback type and allow for registration from individual modules:

```
using DocumentCallback = void (*)(Document*);
std::vector<DocumentCallback> callbacks_;

void RegisterFooCallback(DocumentCallback callback) {
  callbacks_.push_back(callback);
}
```

2. In the implementation, synchronously invoke each callback.

```
if (document) {
  foreach (auto& callback : callbacks_) {
    callback(document);
  }
}
```

A similar pattern can be used in many places to decouple modules/ from classes that will end up in core/ (e.g. ChromeClientImpl, WebViewImpl). Callbacks can be installed in ModulesInitializer::initialize.

WebAXObject.h is a more difficult proposition. It is an export of modules/accessibility/AXObject.h and is used extensively in classes that we would like to move to core/web (e.g. WebNode.cpp, WebViewImpl.cpp, WebDocument.cpp). It's not immediately obvious how to fix this, other than moving accessibility into core/, public/platform/modules or platform/, or moving modules into the same link unit as core. Fixing this is TBD.

## Breaking inverted dependencies.

Once the code has been shuffled into the new locations, we can the start removing various layers that exist only to allow calling from core/ -> web/ (in many cases we just do this when we are doing the initial move of the classes.) From a previous analysis by haraken@ there are over 100 classes that exist just to allow for the inverted dependencies (analysis is old but still relevant).

There are also example bugs that we can resolve by making these changes and fixing the layering.

## Burndown Chart

See this spreadsheet for a list of files to be moved, and progress moving them.