Google

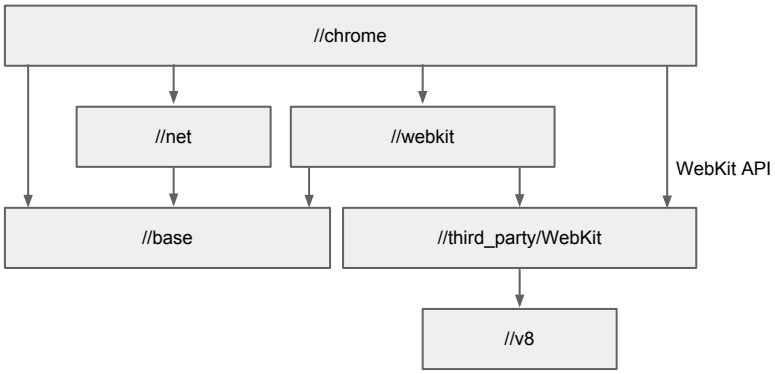
Life of a Browser Component

Rescuing features from a tangled web of dependencies to enable clean reuse

joi@chromium.org

Google

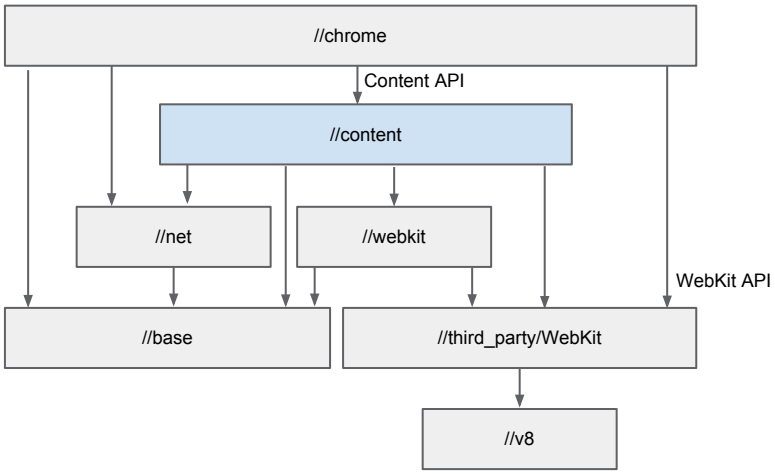
Delicious Layer Cake, v1



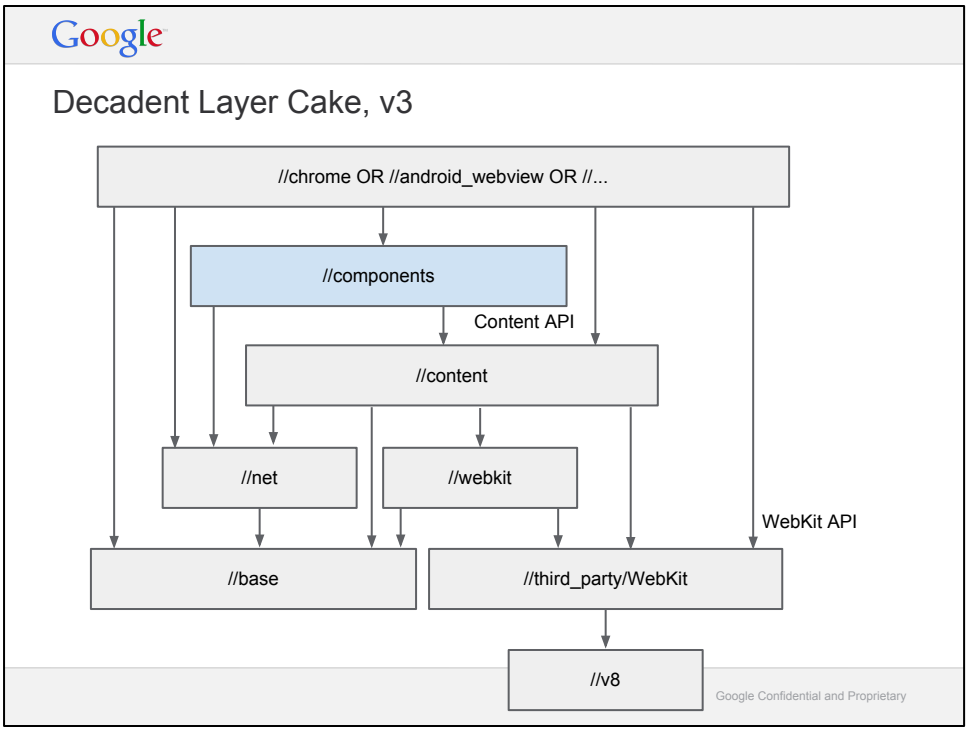
- At some point we had layering that looked like this.
- This is just for illustration, I'm skipping a bunch of things like //ipc, //ui, and others.
- One problem with this model was that the magic goodness of our multi-process, sandboxed architecture was in the same box as all of our user-facing features, so it was a bit too easy to clutter the core architectural pieces with feature-specific code, and it was impossible to reuse the magic goodness.
- Enter jam@, instigating the content module refactoring project.

Google

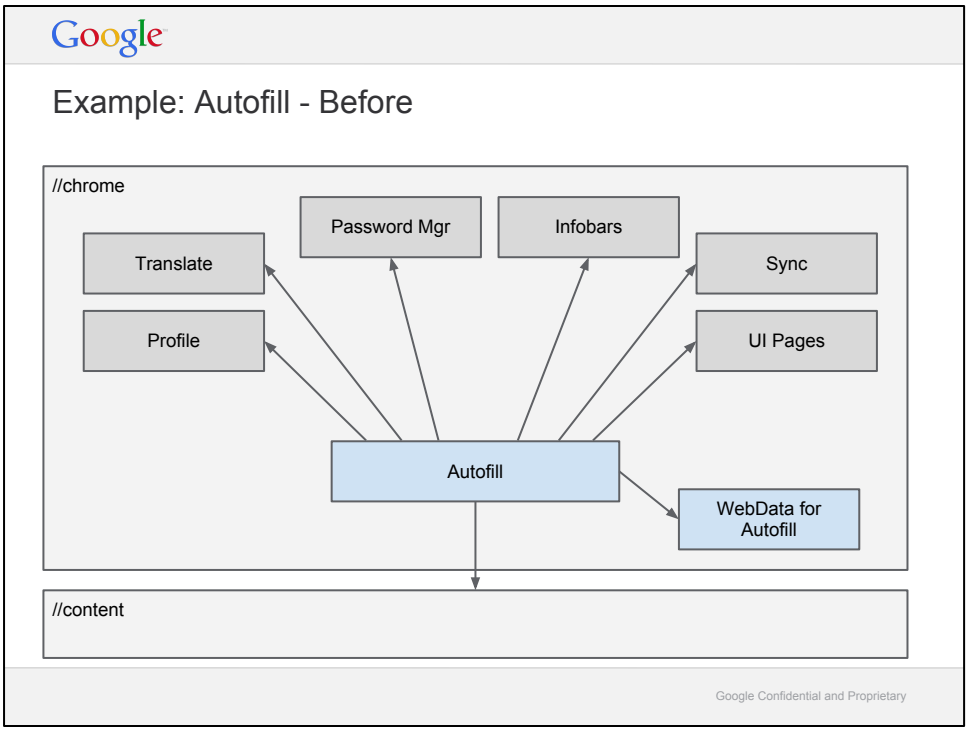
Luscious Layer Cake, v2



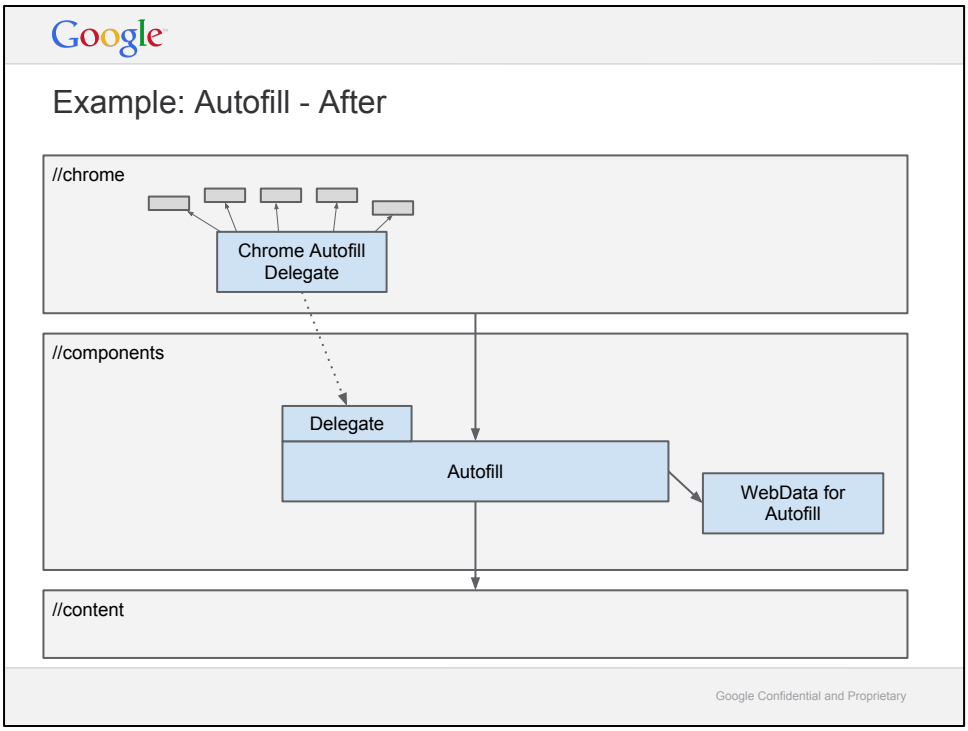
- This is much better - now the architectural bits are in //content and features are in //chrome
- There's a problem there: That //chrome box is desktop Chrome, and also ChromeOS, and also Chrome for Android (through use of #ifdefs). We needed to be able to add new top-level apps that are significantly different from desktop Chrome, e.g. Android WebView, without introducing a huge amount of #ifdefs in //chrome.



- What we've done is introduce another layer, //components, in between //content and //chrome. It is a place for reusable features that can be used by multiple top-level applications



Before componentizing Autofill, it lived in //chrome and depended concretely on an awful lot of Chrome. It also depended on another "lower level" feature, the Autofill-specific part of WebData.



We eliminated all of Autofill's concrete dependencies on parts of Chrome. The Autofill part of WebData was dealt with by also componentizing it. Autofill now defines a delegate interface that any embedder needs to implement. Chrome's implementation of this still has the same dependencies "all over the place" as Autofill did before, but it's much more manageable as it's a tiny piece of code, what you might consider "glue" or "integration" code rather than implementation code.

Google

Example: Autofill - What Moved

//chrome/browser/autofill	→	//components/autofill/browser
//chrome/renderer/autofill	→	//components/autofill/renderer
//chrome/b.../webdata/[generic]	→	//components/webdata
//chrome/b.../webdata/[autofill]	→	//components/autofill/browser/webdata

Concretely, here's how things moved between directories.

Google

What is a Browser Component?

- Lives in //components
- Separate dynamic library in components build
- Generally speaking, a component contains a feature that embedders of //content might want to add
  - Usually business logic, not UI
- Depends only "down"
  - //content and layers below
  - Other components (strictly acyclic)
- Defines embedder interfaces (delegates) to access resources provided by its embedder
- Each embedder creates, owns and configures the component for their use

Google

Structure of a Browser Component

- A component named xyz lives in //components/xyz
- Code should be enclosed in namespace xyz
- Has a strict DEPS file
- A component used only from one process type may contain code directly in its directory
- A more complex component may have a directory per process, e.g.
  - //components/xyz/browser
  - //components/xyz/renderer
  - //components/xyz/common

To start a new Browser Component, just follow the rules from this slide and the previous one.

Extracting an existing feature from //chrome into a component is harder; we have a kind of cookbook for doing it though.

Google

Exfiltration step-by-step

1. Identify a feature you need to extract
2. Work from "leaf nodes"
3. **Write a strict DEPS file with temporary exceptions**
4. **Reduce the set of temporarily-allowed dependencies to zero**
5. Move the code to its //components/xyz directory
6. Fix up .gypi files and add export declarations to build it as a component

Google

DEPS with Temporary Exclusions

```
include_rules = [
  # Sign-in is being componentized.
  "-chrome/common",
  "-chrome/browser",
  "+chrome/browser/signin",

  # TODO(joi): Get this list to zero.
  "!chrome/browser/profiles/profile.h",
  "!chrome/browser/ui/browser_commands.h",
  "!chrome/browser/ui/chrome_pages.h",
]

specific_include_rules = {
  # These files are staying in //chrome.
  r"(chrome_signin_manager_delegate|"
  r"signin_manager_factory)"
  r"\.(h|cc)": [
    "+chrome/browser",
    "+chrome/common",
  ],
}
```


- rules disallow a dependency
+ rules allow it
! rules are like + rules, except they specify something that is like a "temporary +". If somebody adds a new #include of one of these files, they will get a presubmit warning, so they're actively discouraged from working against you when you're componentizing.

Google

Dependency removal cookbook

- **Dependency inversion**
  - This is the fundamental approach; introduce indirection e.g. through delegates, where needed to remove knowledge of the embedder
- **Passing more fundamental objects**, instead of "everything" objects
  - e.g. a PrefService and the SequencedTaskRunner for the IO thread, instead of passing Profile
- **Splitting objects** that bring in tons of dependencies into "core business logic" vs. "integration" or "UI"
  - The integration or UI implementation stays in the embedder, and owns the core business logic, which moves to the component
- ... more in the full cookbook: http://goo.gl/LRgxK ...

Google

Making Componentization Easier

- There is a (small) Browser Components team
- Now focusing less on componentizing features...
  - Working on Sign-in [joi] and Extensions [yoz]
- ...and more on making it easier for others to do so
  - Extracting and generalizing the ProfileKeyedService framework [phajdan, erikwright]
  - Switching to typed notifications (away from NotificationService) [phajdan]
  - Deprecating reference-counted ProfileKeyedServices [caitkp]
  - Rationalizing start-up and shut-down [erikwright]
  - ...and whatever else we can think of that makes it easier to hack the codebase

Google

Call to Action, Q&A

- Writing a new feature?
  ⇒ Make it a Browser Component if it might be reused outside //chrome
- Need to reuse a feature?
  ⇒ Extract it into a Browser Component to reuse it cleanly
- Further documentation on www.chromium.org:
  - Browser Components Design doc http://goo.gl/hhQjL
  - Browser Components Cookbook http://goo.gl/LRgxK
- Need help?
  - browser-components-dev@chromium.org
  - joi@chromium.org