Why we use Google Closure

Hunter Blanks, hblanks@monetate.com / @tildehblanks
github.com/hblanks/talks/

"Something that has been ignored in [previous] responses is the fact that Closure and jQuery are totally different beasts. jQuery is the industry standard way of writing nice, user friendly web pages and user controls. Closure is an application development framework.

Closure comes with a hell of a lot of overheads that are only worthwhile after a certain scale is reached.
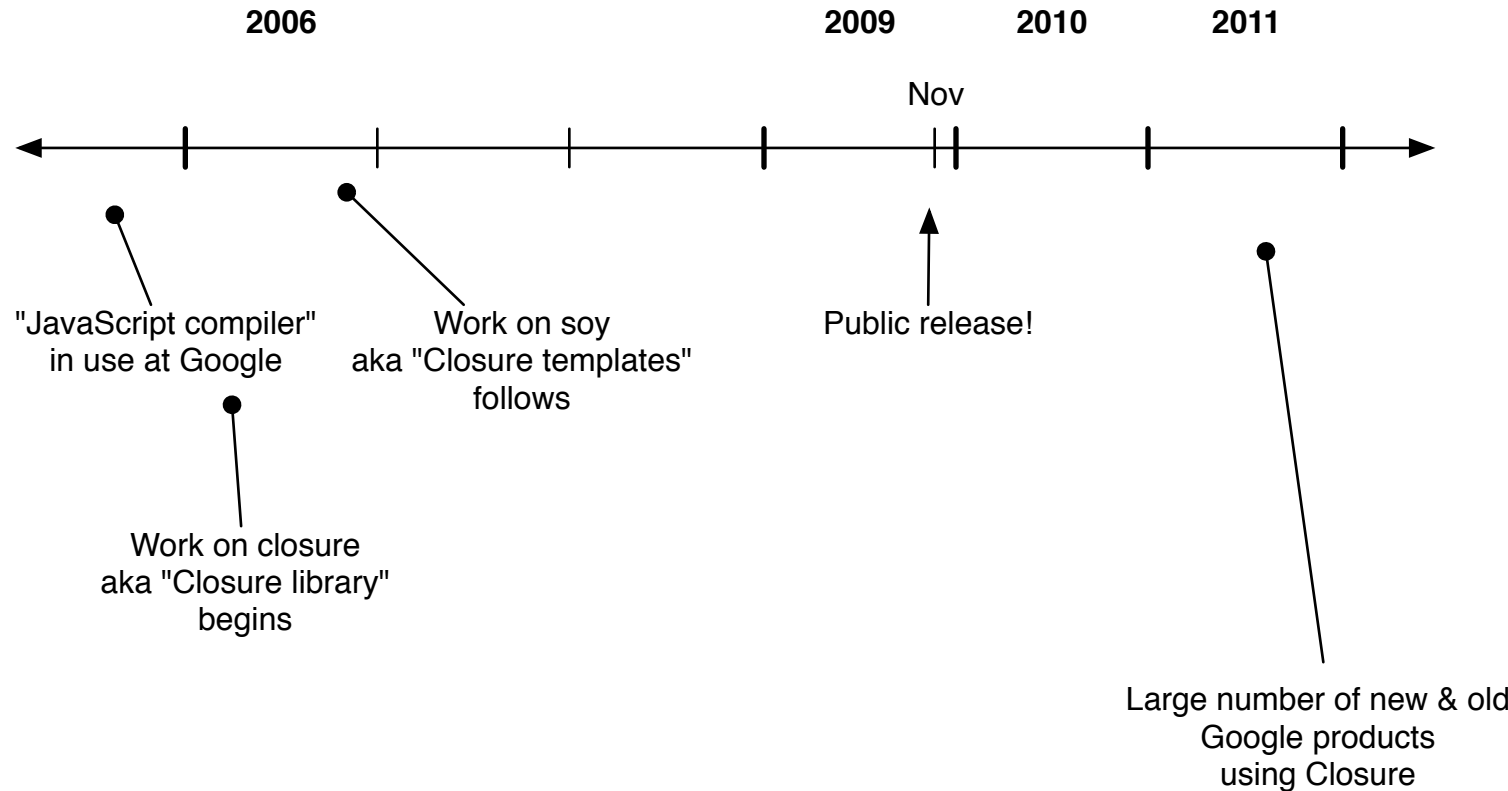
However try writing a complex system (say... 30k+ LOC) in jQuery and you'll feel pain unlike you've never imagined. This is where closure comes into its own (you can always then bring jQuery into a closure project also).

So different tools for different jobs. Both with great support and strong development."

–Guido Tapia

An outsider's timeline of Google Closure

2006   **2009** **2010** **2011**

Nov

"JavaScript compiler"
in use at Google

Work on soy
aka "Closure templates"
follows

Public release!

Work on closure
aka "Closure library"
begins

Large number of new & old
Google products
using Closure

Who uses Closure Library?

The Closure Library serves as the base JavaScript library for many Google products, including

Gmail Maps Docs Sites Books Reader Shopper Calendar Groups Photos

(http://code.google.com/closure/library/)

What is it?

1) **Closure Compiler**. A JavaScript -> Javascript compiler. Key features are:

   - dead code elimination
   - type checking
   - numerous optimizations
   - warnings/errors on unsafe code

2) **Closure Library**. A giant, heavily documented, cross-platform JS library.
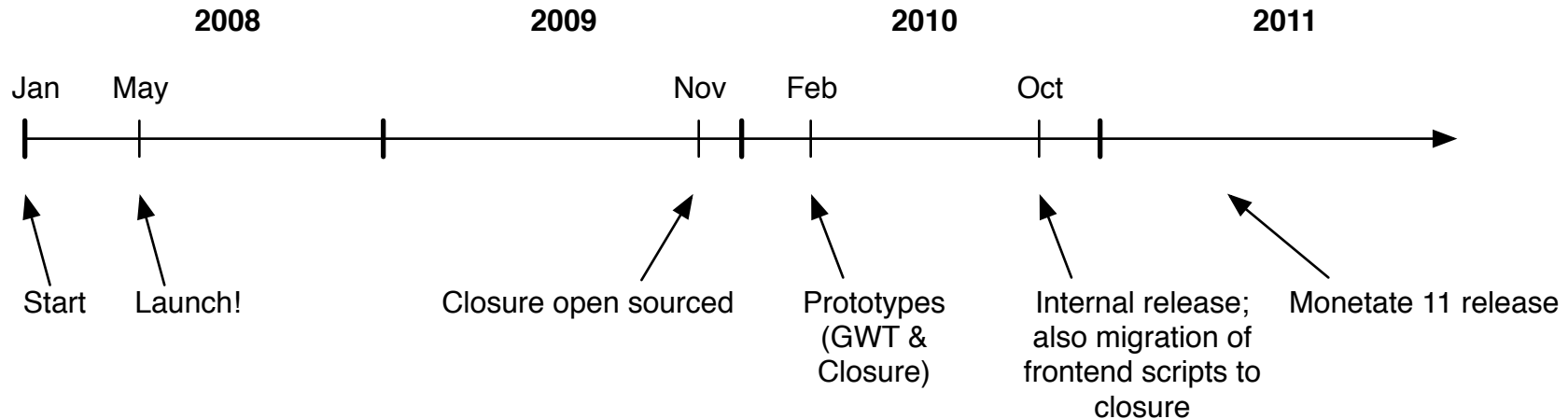   Key features are:

   - cross browser event framework
   - many, many UI components
   - Gmail's rich text editor
   - cross-page & server communication libraries
   - canvas/SVG/VML graphics libraries
   - numerous other libraries (e.g. goog.array, goog.structs.Map, etc.)

3) **Closure templates**. A JS & Java template language/compiler.
   Key features are:

   - HTML structure can be kept separate from your JavaScript
   - you find out about syntax errors and unused varables at compile time!

How did Monetate come to use Google Closure?

Timeline

| 2008 | 2009 | 2010 | 2011 |

Jan May Nov Feb Oct

Start Launch! Closure open sourced Prototypes (GWT & Closure) Internal release; also migration of frontend scripts to closure Monetate 11 release

By late 2008, Monetate had a full-featured, complex web application using Django, jQuery, and traditional AJAX.

By mid 2009, no one wanted to edit the webui.js file, the dozens of different Django template files that it consumed, or the dozens of Django view methods that it talked to.

To build new UI features, and to build the next version of our product, we needed a better framework for building rich web applications.

Later, to maintain our customer-side ("frontend*")* JavaScript code, we also needed a better framework for compiling and binding JavaScript.

Highlights from how we use Closure in Monetate's UI

1) Thanks to Closure templates, we only use Django now to serve JSON-RPC and empty page containers.*Everything else is rendered in the browser.*

HTML ← JavaScript ⇄ JSON-RPC ⇄ Python
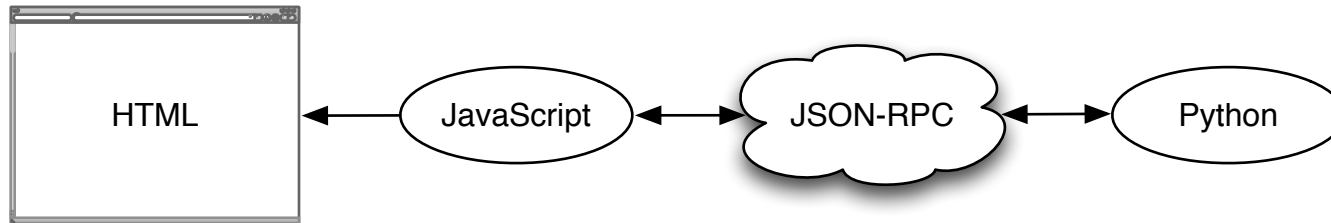
2) Thanks to Closure's Component & EventTarget model, our UI consists of loosely coupled components, which communicate with each other almost always using event buses.

History service ⇄ Presenter
child components → RPC service
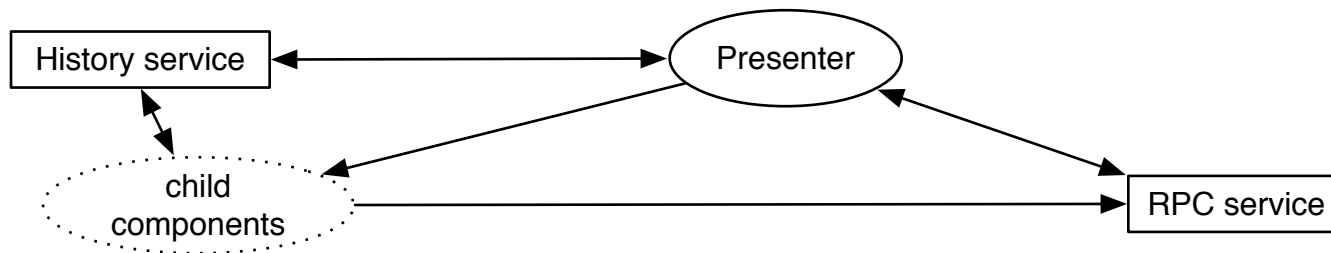
3) Thanks to goog.History, and Ray Ryan's Google IO 2009 talk, URLs are integral to the application — so things like URL cut & paste, the back button, and the refresh button all work as they should.

Highlights from how we use Closure in Monetate's frontend

1) Closure's namespaces let us easily build libraries for common code, and even split out custom code into individual files.

```
/** @fileoverview Monetate lightbox library. */
goog.provide('mc.Lightbox');

goog.require('goog.style');
goog.require('mc.dom');
goog.require('mc.events');
```

2) JSDoc annotations, together with the Closure compiler, make our code more explicit and help us catch *more bugs at compile time.*

```
/**
 * @constructor
 * @extends {goog.Disposable}
 * @param {number} width Width of lightbox.
 * @param {number} height Height of lightbox.
 */
mc.Lightbox = function(width, height) {
    this.width_ = width;
    ...
};
```

3) Among other things, Closure's advanced optimizations remove functions that are never called and inline others. This is key, since for the frontend, small (<16KB) script payload sizes are best.

Final remarks on the bad parts, too

1) You're going to need a build system. It helps to support loading compiled scripts for production and uncompiled scripts (with your own deps.js file) for debugging. Options include

   - plovr
   - closure script
   - make/rake/jake/waf
   - ???

2) Documentation, although rich, is anti-compact. (This is not Python, people.) Michael Bolin's book, *Closure: The Definitive Guide*, is a real help, though.

$\neq$

3) Closure library, although tied to Google's own version control and build system, has no versioned releases, only svn/trunk. So be ready for changes.

Thank you!

Hunter Blanks, hblanks@monetate.com / @tildehblanks
github.com/hblanks/talks/

PS: I'll be pushing a small, demo closure app, with build system,
to github tomorrow. Please check **@tildehblanks** for the announcement.