Contributing to Web Inspector

Devin Rousso

Terminology

- Debuggable
- Target

Debuggable vs Target

$$\begin{tikzcd}
\text{Debuggable} & \text{Target} \\
\bullet\ \texttt{ITML} \arrow[r] & \bullet\ \texttt{ITML} \\
\bullet\ \texttt{JavaScript} \arrow[r] & \bullet\ \texttt{JavaScript} \\
\bullet\ \texttt{Page} \arrow[r] \arrow[dddr] & \bullet\ \texttt{Page} \\
\bullet\ \texttt{ServiceWorker} \arrow[r] & \bullet\ \texttt{ServiceWorker} \\
\bullet\ \texttt{WebPage} \arrow[uur] \arrow[r] \arrow[dr] & \bullet\ \texttt{WebPage} \\
 & \bullet\ \texttt{Worker}
\end{tikzcd}$$

Terminology

- Debuggable
- Target
- Frontend
- Backend
- Protocol
- Remote

Frontend

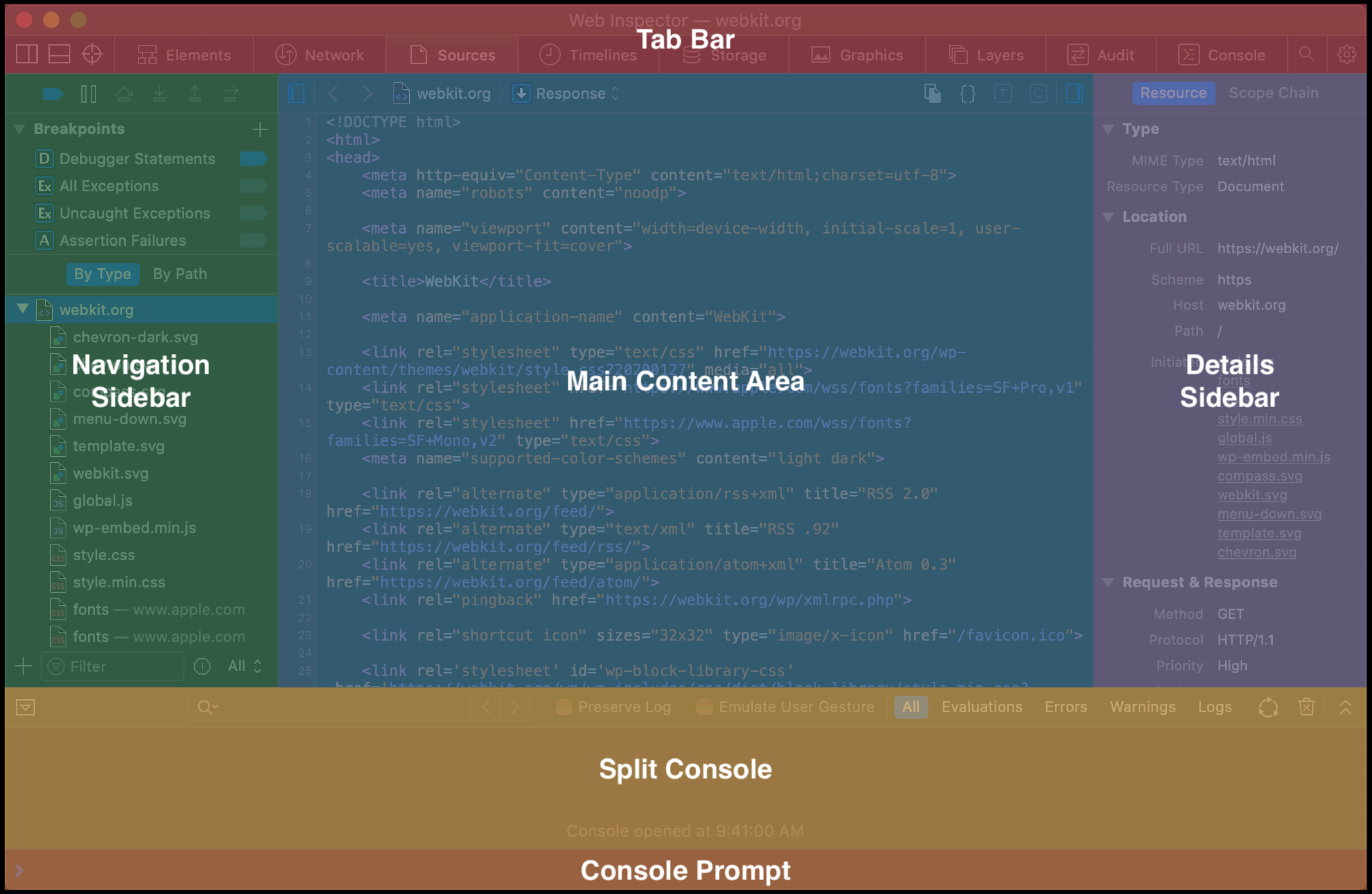
- vanilla HTML+JS+CSS
- `InspectorFrontendHost` vs `InspectorFrontendAPI`
- JS libraries for specific things
  - `CodeMirror` for text editors, `Esprima` for parsing JS, etc.
- event listeners
- custom "layout engine"
- MVC pattern

Frontend
MVC

- controllers mainly in the form of `Manager`
  - one-to-many relationship of `Manager` to `Target`
- mostly model and view
  - models are usually a representation of something in the Protocol

Frontend

View/UI



```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
    <meta name="robots" content="noodp">

    <meta name="viewport" content="width=device-width, initial-scale=1, user-
scalable=yes, viewport-fit=cover">

    <title>WebKit</title>

    <meta name="application-name" content="WebKit">

    <link rel="stylesheet" type="text/css" href="https://webkit.org/wp-
content/themes/webkit/style.css?20200127" media="all">
    <link rel="stylesheet" href="https://www.apple.com/wss/fonts?families=SF+Pro,v1"
type="text/css">
    <link rel="stylesheet" href="https://www.apple.com/wss/fonts?
families=SF+Mono,v2" type="text/css">
    <meta name="supported-color-schemes" content="light dark">

    <link rel="alternate" type="application/rss+xml" title="RSS 2.0"
href="https://webkit.org/feed/">
    <link rel="alternate" type="text/xml" title="RSS .92"
href="https://webkit.org/feed/rss/">
    <link rel="alternate" type="application/atom+xml" title="Atom 0.3"
href="https://webkit.org/feed/atom/">
    <link rel="pingback" href="https://webkit.org/wp/xmlrpc.php">

    <link rel="shortcut icon" sizes="32x32" type="image/x-icon" href="/favicon.ico">

    <link rel='stylesheet' id='wp-block-library-css'
```

Type

MIME Type	text/html
Resource Type	Document

Location

Full URL	https://webkit.org/
Scheme	https
Host	webkit.org
Path	/
Initia	fonts style.min.css global.js wp-embed.min.js compass.svg webkit.svg menu-down.svg template.svg chevron.svg

Request & Response

Method	GET
Protocol	HTTP/1.1
Priority	High

Frontend

View/UI Components

- `WI.TreeOutline` and `WI.TreeElement`
- `WI.Table`
- `WI.DetailsSection` et al
- `WI.NavigationBar` and `WI.NavigationItem`
- `WI.Popover`
- etc

Protocol

- JSON-RPC
- Domains

Protocol

Domains

- `Animation`
- `ApplicationCache`
- `Audit`
- `Browser`
- `Canvas`
- `Console`
- `CPUProfiler`
- `CSS`
- `Database`
- `Debugger`
- `DOM`
- `DOMDebugger`
- `DOMStorage`
- `GenericTypes`
- `Heap`
- `IndexedDB`
- `Inspector`
- `LayerTree`
- `Memory`
- `Network`
- `Page`
- `Recording`
- `Runtime`
- `ScriptProfiler`
- `Security`
- `ServiceWorker`
- `Target`
- `Timeline`
- `Worker`

Protocol

- JSON-RPC
- Domains
- Types
- Commands
- Events

Protocol

JSON

Protocol

JSON

```
{
    "domain": "DOM",
    "debuggableTypes": ["itml", "page", "web-page"],
    "targetTypes": ["itml", "page"],
    "types": [...],
    "commands": [...],
    "events": [...]
}
```

Protocol

JSON types

```
{
    "types": [
        {
            "id": "NodeId",
            "type": "integer",
            "description": "Unique DOM node identifier."
        },
    ],
}
```

Protocol

JSON types

```
{
    "types": [
        {
            "id": "PseudoType",
            "type": "string",
            "enum": ["before", "after"],
            "description": "Pseudo element type."
        },
    ],
}
```

Protocol

JSON types

```
{
    "types": [
        {
            "id": "Node",
            "type": "object",
            "properties": [
                { "name": "nodeId", "$ref": "NodeId" },
                { "name": "nodeType", "type": "integer" },
            ]
        },
    ],
}
```

Protocol

JSON commands

```
{
    "commands": [
        {
            "name": "getDocument",
            "description": "Returns the root DOM node.",
            "returns": [
                { "name": "root", "$ref": "Node" }
            ]
        },
    ],
}
```

Protocol

JSON commands

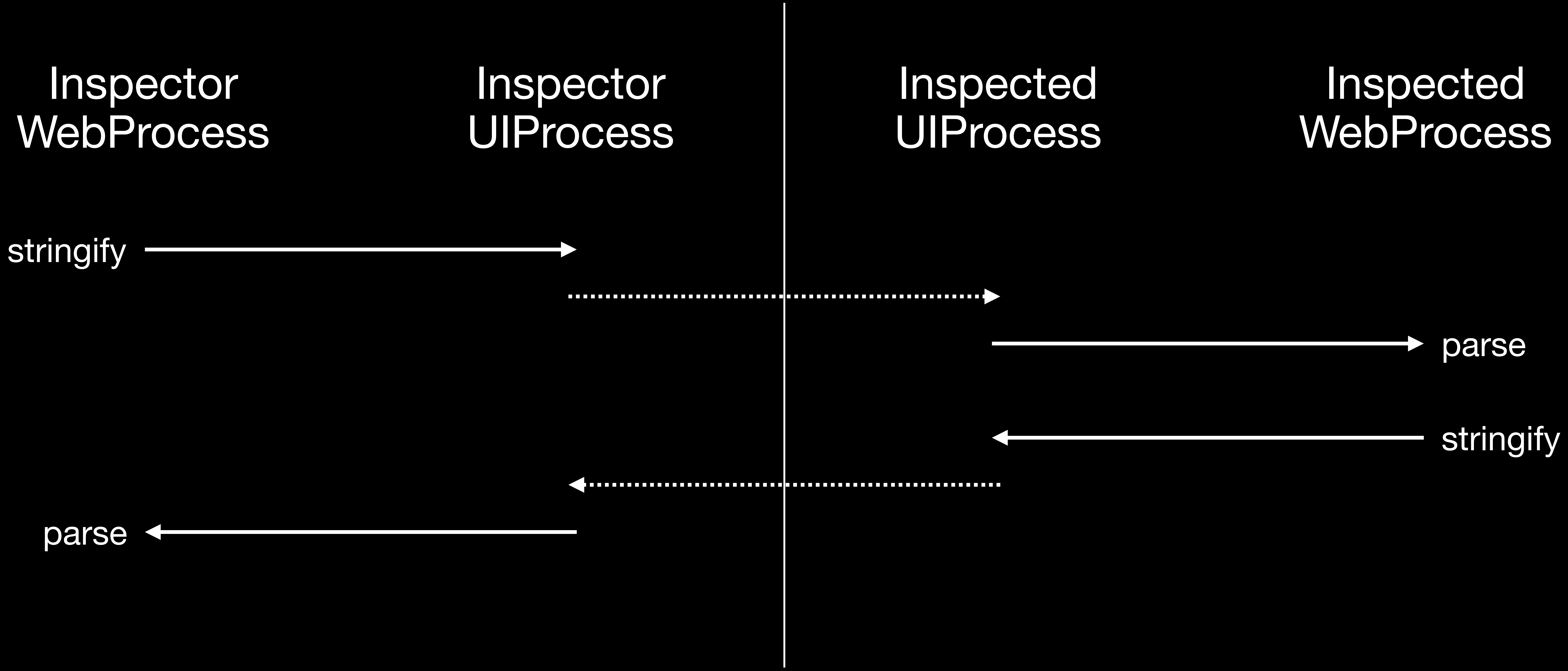
```
{
    "commands": [
        {
            "name": "setNodeName",
            "description": "Sets node name for a node with given id.",
            "targetTypes": ["page"],
            "parameters": [
                { "name": "nodeId", "$ref": "NodeId" },
                { "name": "name", "type": "string"  }
            ],
            "returns": [
                { "name": "nodeId", "$ref": "NodeId" }
            ]
        },
    ],
}
```

Protocol

JSON events

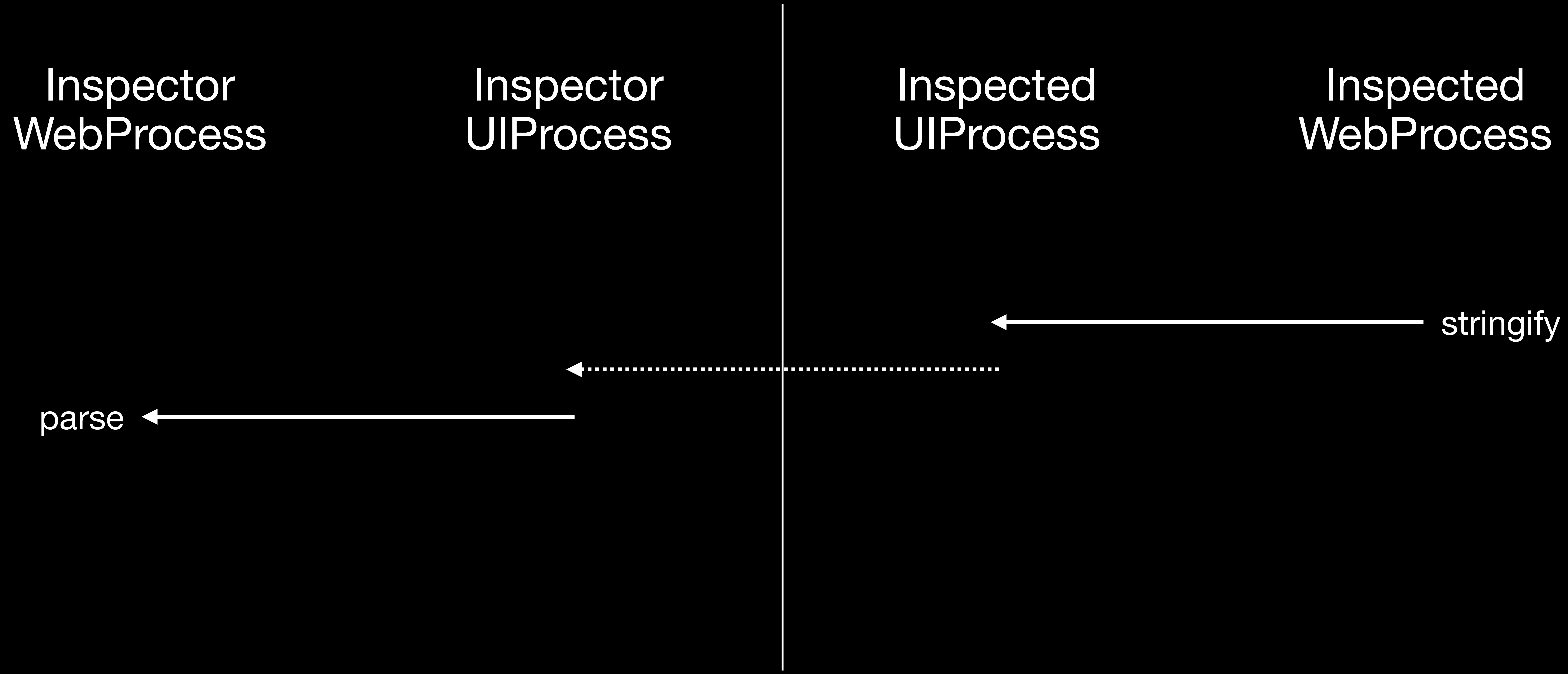
```
{
    "events": [
        {
            "name": "attributeModified",
            "parameters": [
                { "name": "nodeId", "$ref": "NodeId" },
                { "name": "name", "type": "string" },
                { "name": "value", "type": "string" }
            ]
        },
    ]
}
```

Protocol
command



Protocol

event



Protocol

- compatibility from final shipped copy of the protocol for each macOS and iOS
  - `if (InspectorBackend.hasCommand("Debugger.stepNext")) {`
- relevant code is autogenerated from protocol for JS and C++
  - `target.DOMAgent.setNodeName(nodeId, name).then(({nodeId}) => { ... })`
  - `Inspector::Protocol::ErrorStringOr<int /* nodeId */> setNodeName(int nodeId, const String& name)`
  - `void attributeModified(int nodeId, const String& name, const String& value);`
- heavy usage of `WTF::JSON`

Backend

- each debuggable has a `Controller`
- each domain has an `Agent` (per target)
  - prefixed by target (e.g. `InspectorDebuggerAgent` (base) vs `PageDebuggerAgent` vs `WorkerDebuggerAgent`)
  - keeps Web Inspector logic, data, etc. contained
- in JavaScriptCore, go through the `Debugger`
- in WebCore, use `InspectorInstrumentation`

General Tips

- lots of prior art for all sorts of things
  - changes usually touch everything (i.e. frontend, protocol, and backend)
- use Web Inspector to debug Web Inspector (a.k.a. inspector^2)
- ESLint is your friend in the frontend
- protocol and logic tests only (i.e. no UI tests)

Q/A