Contributing to Web Inspector

Devin Rousso
Terminology

• Debuggable
• Target
Debuggable vs Target

Debuggable
- ITML
- JavaScript
- Page
- ServiceWorker
- WebPage

Target
- ITML
- JavaScript
- Page
- ServiceWorker
- WebPage
- Worker
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
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
<table>
<thead>
<tr>
<th>Protocol Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animation</td>
</tr>
<tr>
<td>ApplicationCache</td>
</tr>
<tr>
<td>Audit</td>
</tr>
<tr>
<td>Browser</td>
</tr>
<tr>
<td>Canvas</td>
</tr>
<tr>
<td>Console</td>
</tr>
<tr>
<td>CPUProfiler</td>
</tr>
<tr>
<td>CSS</td>
</tr>
<tr>
<td>Database</td>
</tr>
<tr>
<td>Debugger</td>
</tr>
<tr>
<td>DOM</td>
</tr>
<tr>
<td>DOMDebugger</td>
</tr>
<tr>
<td>DOMStorage</td>
</tr>
<tr>
<td>GenericTypes</td>
</tr>
<tr>
<td>Heap</td>
</tr>
<tr>
<td>IndexedDB</td>
</tr>
<tr>
<td>Inspector</td>
</tr>
<tr>
<td>LayerTree</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Network</td>
</tr>
<tr>
<td>Page</td>
</tr>
<tr>
<td>Recording</td>
</tr>
<tr>
<td>Runtime</td>
</tr>
<tr>
<td>ScriptProfiler</td>
</tr>
<tr>
<td>Security</td>
</tr>
<tr>
<td>ServiceWorker</td>
</tr>
<tr>
<td>Target</td>
</tr>
<tr>
<td>Timeline</td>
</tr>
<tr>
<td>Worker</td>
</tr>
</tbody>
</table>
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

```json
{
   "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."
        },
        {
            "name": "root",
            "$ref": "Node"
        }
    ]
}
Protocol

JSON commands

```json
{
    "commands": [
        {
            "name": "setName",
            "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

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

Inspector WebProcess  Inspector UIProcess  Inspected UIProcess  Inspected WebProcess

stringify  parse  

parse  stringify  parse
Protocol

event

Inspector
WebProcess

Inspector
UIProcess

Inspected
UIProcess

Inspected
WebProcess

parse

stringify
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.setName(nodeId, name).then(({nodeId}) => { ... })

• Inspector::Protocol::ErrorStringOr<int /* nodeId */) setName(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)