Team Rocket

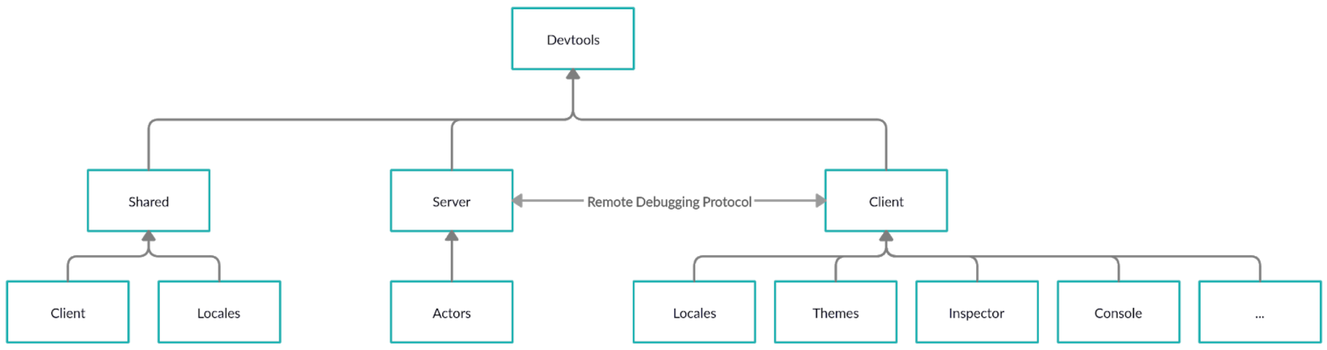
CSCD01 Deliverable #1

February 26th, 2020

Team Members: Yasir Haque, Adnan Shahid, Patricia Lee,
Sol Han, and Jacqueline Chan

Table of Contents

Firefox System Architecture



General Overview

The central Mozilla Firefox browser is made up of several robust component projects to which developers may contribute. Our group will be primarily focusing on their Firefox Dev Tools web application as it allows us to leverage our prior skills in full stack/web development.

The Firefox Dev Tools web app can be broken down into three major parts which include the client-side, server, and shared resources. The client-side consists of multiple constituents such as the inspector, console, debugger, etc. The project uses React.js as its framework of choice to create the UI interface. React is ideal due to its component design philosophy which allows for great scalability for an application as large as this. Furthermore, its inability to manage states is addressed with the use of Redux. Through the use of Redux, components can now communicate states to one another where they would normally have been unable to.

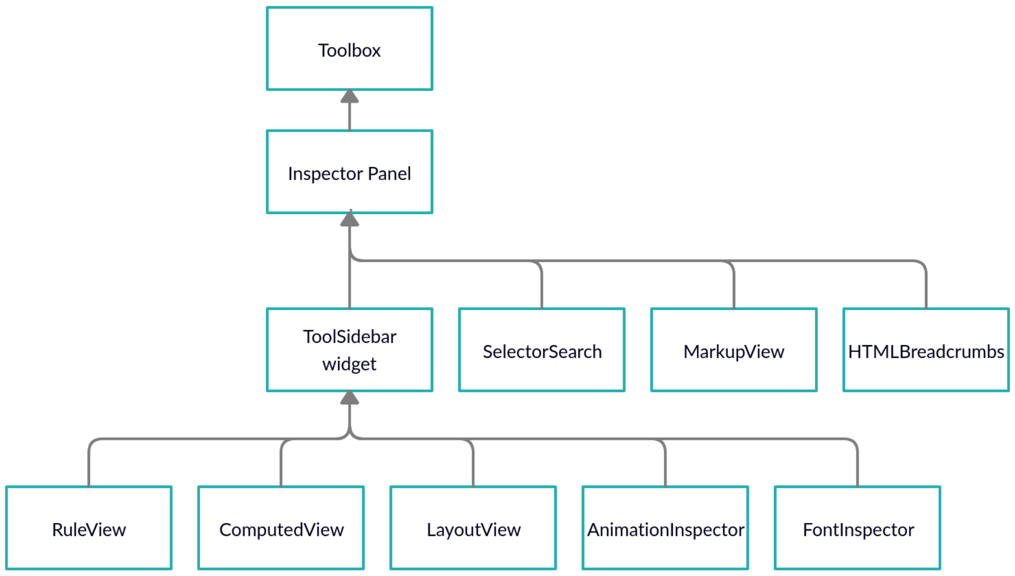
The server-side of the project contains actors whose function is to trade JSON packets with the client. A debugger may be working with tabs and other browser tools which would each have an individual actor representing them in the backend. The Remote Debugging Protocol is what allows this communication between frontend components and actors to occur. The protocol provides information regarding DOM nodes, CSS rules, etc. used in client-side apps and can be extended for use in various client-server relationships.

Lastly, the “shared” folder contains code that both the client and server side of the application shares. It involves several external libraries that are used throughout the project.

As mentioned, the client side of the DevTools application is made up of multiple tool components. The DevTools class is responsible for registering, unregistering, and accessing developer tools, toolboxes and themes in Firefox. Registered tools will have their own tab located within a box, called the Toolbox. The Toolbox and the tools contained within it are linked to a Target, which is the object being debugged. The Target would normally be a tab or web page, but it can also be a window. Given the multiple components found within the Toolbox, this report is focused on the inspector panel, highlighter tools, and console, as concrete examples to help acclimate to the application's overall structure.

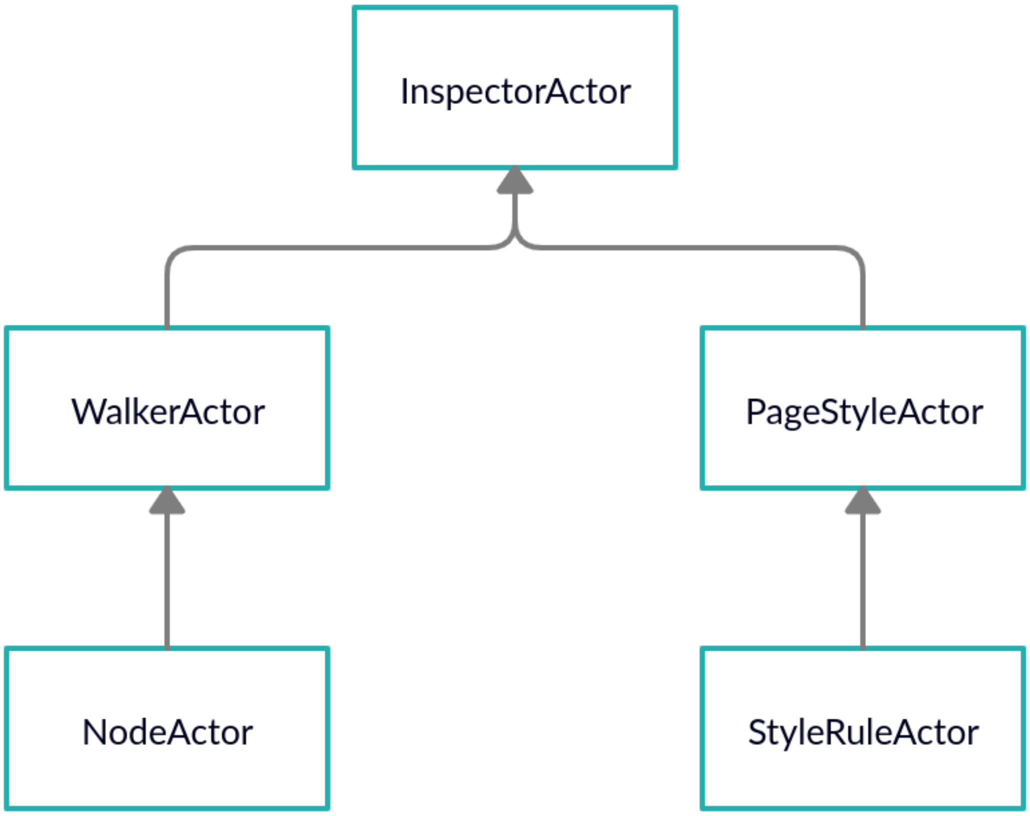
Inspector Overview

The inspector panel is one of many tabs inside the toolbox. It allows the user to select, manipulate and investigate various DOM elements within the webpage. It consists of several views that are in charge of previewing html, showing CSS rules, searching through elements and more. The front end layout structure is as follows:



See Appendix A for UML diagrams outlining the three major panel features SelectorSearch, MarkupView, and HTMLBreadcrumbs.

When the toolbox is initially opened, the server creates actors in the backend which panels such as inspector rely upon to operate. These actors enable panels to do things such as preview, link to, and output DOM nodes. The layout of the actors on the server is as follows:

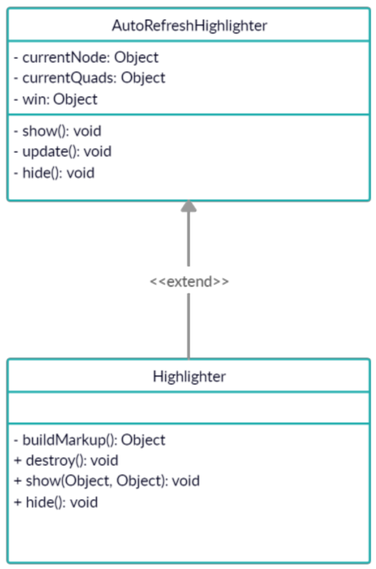


See Appendix B for UML diagrams outlining additional actors within the server
side of the application.

The inspector panel uses the inspector actor to acquire the walker and pagestyle actors respectively. The walker actor is used as the foundation to help traverse the DOM on the page, iframes, pseudo elements and more. They provide trees of node actor objects to do so. The pagestyle actor and style rule actors send information regarding the styling of individual elements to the panel as they are traversed.

Highlighter Overview

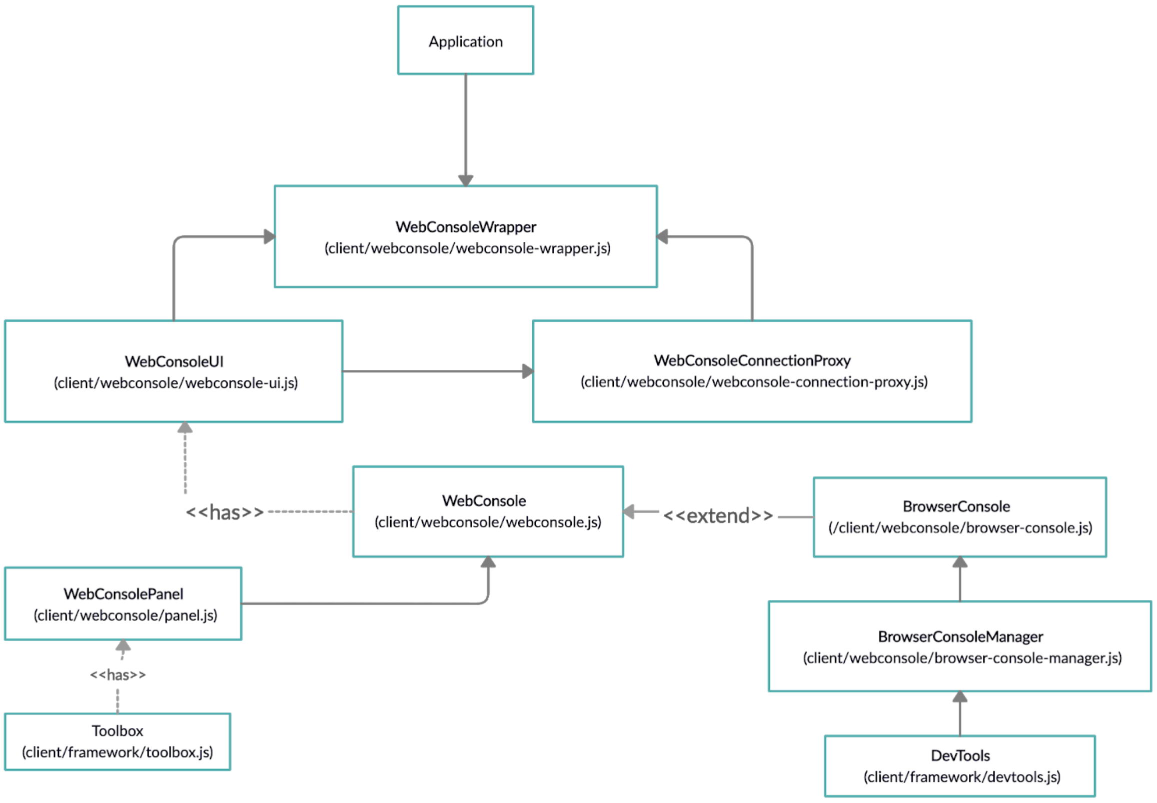
Highlighters are what the DevTools will showcase on a page to emphasize DOM elements, box models, etc. Rather than manipulating the DOM and being instantiated client side, these highlighters are actually formed in the backend to allow various types clients (i.e. a remote device) to use them. They make use of the Remote Debugging Protocol (as explained in the general overview) and have their own API to function.



When highlighters are formed to highlight elements in the DOM, they generally inherit from a parent class known AutoRefreshHighlighter. This parent class constantly updates to check whether the highlighted element has changed in terms shape (such as through HTML manipulation) to draw on top of the appropriate areas. The objects returned by the buildMarkup() function within the Highlighter class are the containers which indicate the DOM element to be highlighted. The elements returned in these containers can be shown, hidden, resized, and more through the use of highlighter's API.

The Console Tool

The Web Console is where all the logs from a particular page are rendered. This would include network requests, security errors, warnings, and information logged by Javascript code. There would be a link beside the messages which would redirect users to the line of code that generated the log. Additionally, there is a command line feature that allows users to type in Javascript code with respect to the context of the page.


Application
WebConsoleWrapper
(client/webconsole/webconsole-wrapper.js)
WebConsoleUI
(client/webconsole/webconsole-ui.js)
WebConsoleConnectionProxy
(client/webconsole/webconsole-connection-proxy.js)
<<has>>
WebConsole
(client/webconsole/webconsole.js)
<<extend>>
BrowserConsole
(/client/webconsole/browser-console.js)
WebConsolePanel
(client/webconsole/panel.js)
BrowserConsoleManager
(client/webconsole/browser-console-manager.js)
<<has>>
Toolbox
(client/framework/toolbox.js)
DevTools
(client/framework/devtools.js)

Please See Appendix C for the UML diagrams outlining a more detailed view of the Web Console component.

From the bottom, we first have the Toolbox class that has panels from each component, Web Console (panel.js) being one of them. The console will be able to be launched inside of the DevTool toolbox by the WebConsolePanel, which controls/uses the WebConsole class. BrowserConsole extends WebConsole and is used by another framework (devtools.js). We then have that the WebConsoleUI uses WebConsoleConnectionProxy and WebConsoleWrapper while at the same time, WebConsoleConnectionProxy calls methods from the WebConsoleWrapper. Ultimately, it is the the WebConsoleWrapper that renders the (React) application.

One interesting aspect would be the inheritance design pattern of the Browser Console extending from the Web Console class. The Browser Console is similar to the web console except that instead of having the console features only apply to one single tab, it applies to the whole browser instead. For example, the logs such as network requests, errors and warnings relate to all the content in the tabs. JavaScript code executed in the command line would then apply to the whole browser scope. Typically in CSCC01, we would have classes extend from a more generic base class, i.e. the Animal class would be the base class and Cats would be extending from it. The Browser Console, solely based on the description, may lead people to think that it is the base case since it has such a wide scope. Therefore, many might associate that as being more generic since the Web Console only applies to one tab. However, here we see that they are treating Web Console as the base class instead and it is a more generic class because they have designed with having a greater scope as an extra feature instead. Digging through the code, this seems to be a smart implementation as only a few functions and lists were needed in the Browser Console class to extend the Web Console functionalities to the whole browser. Also, it is noted that the Browser Console tool was a feature that was later introduced after the Web Console (according to documentation). Therefore, this suggests that the architecture for this console component was well thought out and maintainable since it can easily scale to handle additional console instances.

Software Development Process

Kanban

The software development process that we'll be using is Kanban. This will be done using the tool Trello. We have chosen to do this as it is visually appealing and easy to understand for all team members, while also being free. The Kanban board allows us to easily depict how much progress we are making on issues and identify any roadblocks that are hindering tasks. Having daily "stand-ups" also gives us an opportunity to vocalize our progress with one another to prevent any silos within the team. In these stand-up meetings, we will decide who will be assigned new tasks, who requires extra time or help, and plan out how we will meet our deadlines. By focusing on continuous development, Kanban can help maximize the productivity of our team by reducing idle time as all team members can easily view the status of the project with our Kanban board.

One modification we made to the Kanban development process is choosing our own columns, which consist of: To Do, In Progress, Test, Review, Done and Blocked. We believe this selection of columns will help streamline our development process. We have also set WIP (Work In Progress) limits to the columns on the board. The In Progress column has a limit of 4 tasks, the Test column has a limit of 3 tasks, and the Review column has a limit of 2 tasks. We are enforcing these limits because we want to ensure that our workflow is continuous. Our group collectively agreed that testing and reviewing is something that we aren't all that fond of and therefore we gave a lower limit so that it forces us to finish these tasks before moving on to take on new ones. If there is ever a situation where a task is complete, but cannot be moved to its next column due a WIP limit, the card will be moved to the Blocked column. We will also use the Blocked column for situations where we lack sufficient knowledge on a task or other extraneous circumstances that are preventing us from completing a task at that point in time.

Another modification is the use of story points in the Kanban process. Story points are used for estimating the overall effort required to complete a backlog item. For this project, our team has decided that each story point will represent 1 hour of work. Story points will be added to tasks on the Trello board using an Agile Tool add on. The use of story points will help us identify the larger tasks that may require multiple assignees. Story points for tasks will be estimated using Planning Poker.

Pros and Cons of other Software Processes

Waterfall

The Waterfall methodology is a plan-driven process that has separate distinct phases for specification and development. It can be very useful because it is simple to use and easy to understand with its intuitive progression through its stages. We chose to use Kanban over Waterfall mainly because of Kansan's flexibility. The Waterfall model follows a strict sequence of stages and is quite inflexible when faced with unexpected changes. In addition, the testing phase in the Waterfall model is one of the last stages and since the model progresses one stage at a time, testing will be delayed. In contrast, with Kanban, the testing stage can be worked on concurrently with the other stages, which can help us complete tasks more efficiently

Scrum

Scrum is an Agile process that focuses on managing iterative development. A benefit of Scrum is that work is done simultaneously rather than sequentially, thus making the process more flexible and adaptable to changing requirements. However, Scrum involves many meetings including daily stand-up meetings and Sprint reviews, which can take up considerable time and resources. Another key problem is that the process involves arranging meetings with a Product Owner, who represents the users and customers of the product. They would be responsible for defining and prioritizing the team's backlog items, however in our situation, we do not have any clients to fill to role as the Product Owner, thus it is not suitable for our current project.

Extreme Programming (XP)

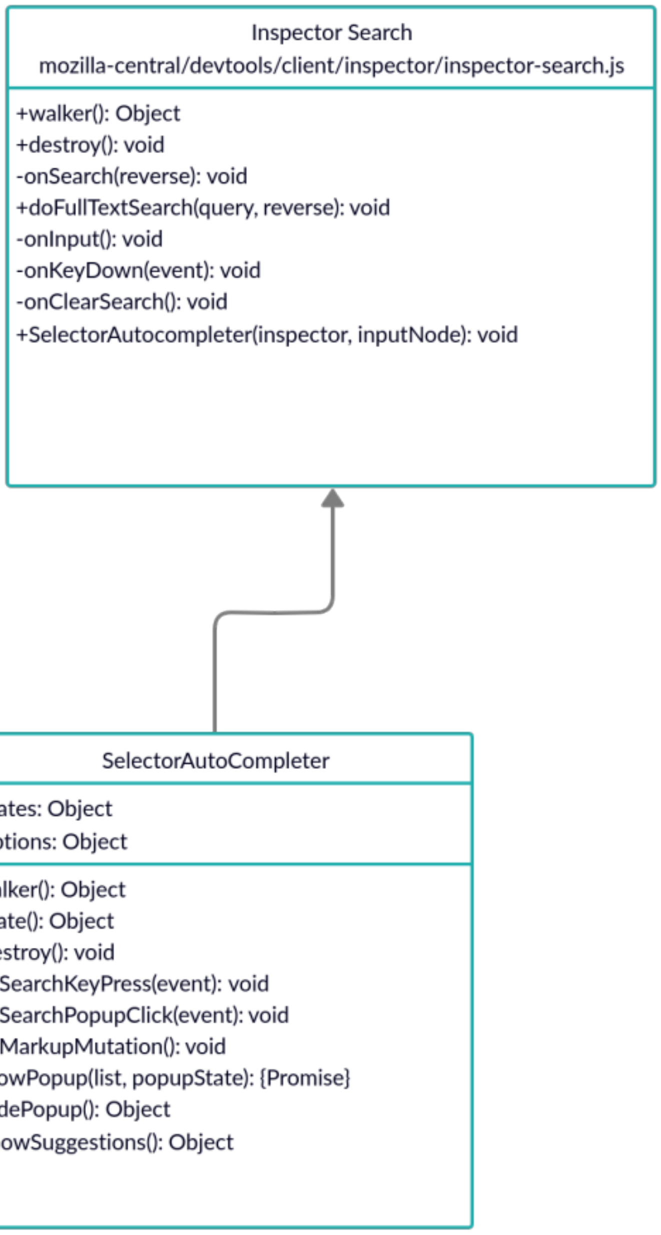
Extreme programming is another Agile methodology that involves iterative development, daily meetings, and release and iteration planning. The aim of this framework is to produce higher quality software with small and frequent releases. Extreme programming can be beneficial with its fast-paced working environment because it results in work getting finished fast and deployed fast. Unfortunately, Extreme programming would not be suitable for our team and this project because we all have other courses that require our time and resources, so meeting its tight deadlines would be tough and stressful. Also, similar to Scrum, XP requires customer involvement. However in our situation, we do not have customers to consult.

Appendix

Appendix A

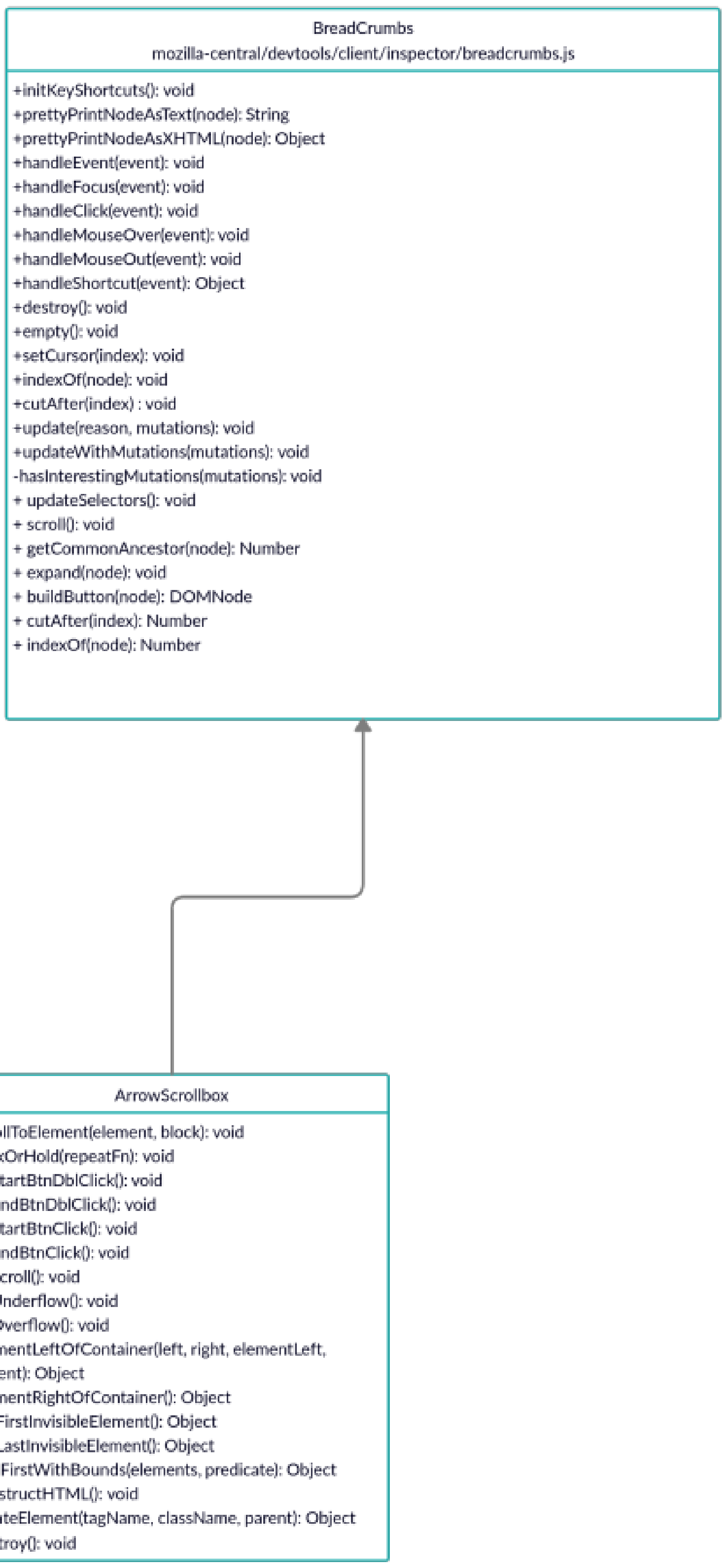
Appendix A i.

- The frontend component in charge querying the DOM for elements. Creates an autocompleter which looks at the DOM structure to aid in searches/suggestions.


Inspector Search
mozilla-central/devtools/client/inspector/inspector-search.js
+walker(): Object
+destroy(): void
-onSearch(reverse): void
+doFullTextSearch(query, reverse): void
-onInput(): void
-onKeyDown(event): void
-onClearSearch(): void
+SelectorAutocompleter(inspector, inputNode): void
SelectorAutoCompleter
- states: Object
- options: Object
-walker(): Object
+state(): Object
+destroy(): void
-onSearchKeyPress(event): void
-onSearchPopupClick(event): void
-onMarkupMutation(): void
-showPopup(list, popupState): {Promise}
+hidePopup(): Object
+showSuggestions(): Object

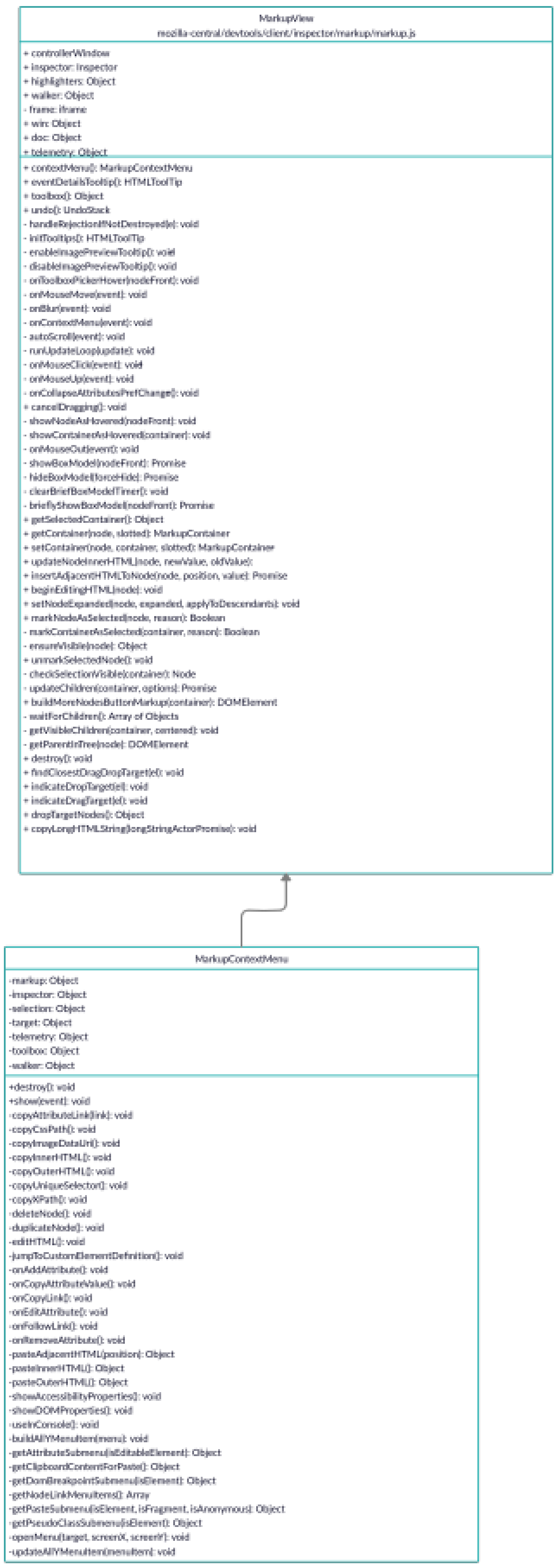
Appendix A ii.

- Frontend component within the inspector panel that renders more information after clicking “breadcrumbs” within the markup html viewer. Creates an instance of an arrow scroll box to help render more and potentially overflowing html information.

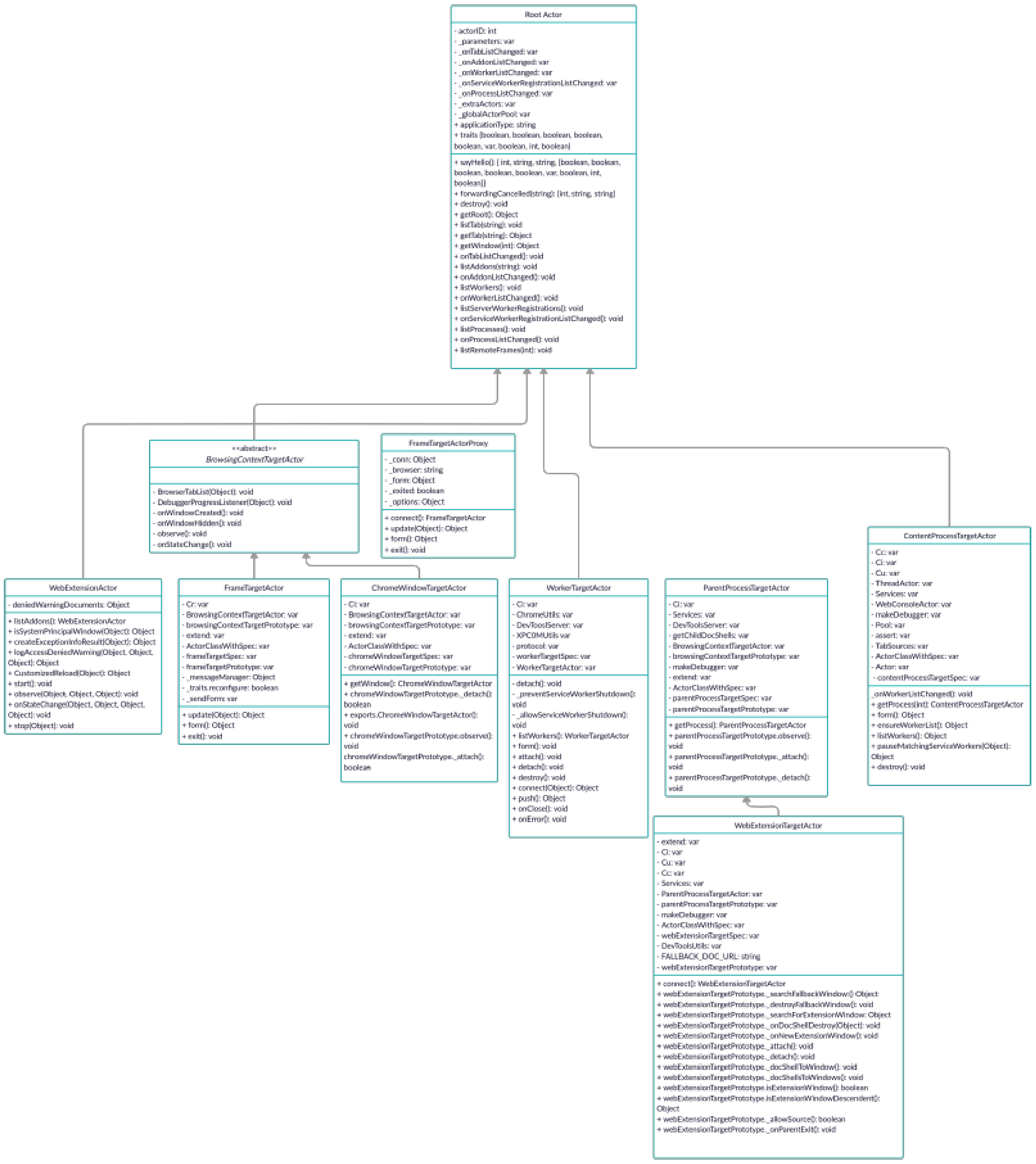

BreadCrumbs
mozilla-central/devtools/client/inspector/breadcrumbs.js
+initKeyShortcuts(): void
+prettyPrintNodeAsText(node): String
+prettyPrintNodeAsXHTML(node): Object
+handleEvent(event): void
+handleFocus(event): void
+handleClick(event): void
+handleMouseOver(event): void
+handleMouseOut(event): void
+handleShortcut(event): Object
+destroy(): void
+empty(): void
+setCursor(index): void
+indexOf(node): void
+cutAfter(index) : void
+update(reason, mutations): void
+updateWithMutations(mutations): void
-hasInterestingMutations(mutations): void
+ updateSelectors(): void
+ scroll(): void
+ getCommonAncestor(node): Number
+ expand(node): void
+ buildButton(node): DOMNode
+ cutAfter(index): Number
+ indexOf(node): Number
ArrowScrollbox
+scrollToElement(element, block): void
+clickOrHold(repeatFn): void
+onStartBtnDblClick(): void
+onEndBtnDblClick(): void
+onStartBtnClick(): void
+onEndBtnClick(): void
+onScroll(): void
+onUnderflow(): void
+onOverflow(): void
+elementLeftOfContainer(left, right, elementLeft, element): Object
+elementRightOfContainer(): Object
+getFirstInvisibleElement(): Object
+getLastInvisibleElement(): Object
+findFirstWithBounds(elements, predicate): Object
+constructHTML(): void
+createElement(tagName, className, parent): Object
+destroy(): void

Appendix A iii.

- The frontend component within the inspector panel that allows the user to view the HTML rendered on the page. The MarkUp view creates an instance of a MarkUpContextMenu which allows the user to perform additional features such as edit HTML, copy HTML paths, duplicate nodes and various other feats that one might used to in their standard developer console.


MarkupView
mozilla-central/devtools/client/inspector/markup/markup.js
+ controllerWindow
+ inspector: Inspector
+ highlighters: Object
+ walker: Object
- frame: iframe
+ win: Object
+ doc: Object
+ telemetry: Object
+ contextMenu(): MarkupContextMenu
+ eventDetailsTooltip(): HTMLToolTip
+ toolbox(): Object
+ undo(): UndoStack
- handleRejectionIfNotDestroyed(e): void
- initTooltips(): HTMLToolTip
- enableImagePreviewTooltip(): void
- disableImagePreviewTooltip(): void
- onToolboxPickerHover(nodeFront): void
- onMouseMove(event): void
- onBlur(event): void
- onContextMenu(event): void
- autoScroll(event): void
- runUpdateLoop(update): void
- onMouseClick(event): void
- onMouseUp(event): void
- onCollapseAttributesPrefChange(): void
+ cancelDragging(): void
- showNodeAsHovered(nodeFront): void
- showContainerAsHovered(container): void
- onMouseOut(event): void
- showBoxModel(nodeFront): Promise
- hideBoxModel(forceHide): Promise
- clearBriefBoxModelTimer(): void
- brieflyShowBoxModel(nodeFront): Promise
+ getSelectedContainer(): Object
+ getContainer(node, slotted): MarkupContainer
+ setContainer(node, container, slotted): MarkupContainer
+ updateNodeInnerHTML(node, newValue, oldValue):
+ insertAdjacentHTMLToNode(node, position, value): Promise
+ beginEditingHTML(node): void
+ setNodeExpanded(node, expanded, applyToDescendants): void
+ markNodeAsSelected(node, reason): Boolean
- markContainerAsSelected(container, reason): Boolean
- ensureVisible(node): Object
+ unmarkSelectedNode(): void
- checkSelectionVisible(container): Node
- updateChildren(container, options): Promise
+ buildMoreNodesButtonMarkup(container): DOMElement
- waitForChildren(): Array of Objects
- getVisibleChildren(container, centered): void
- getParentInTree(node): DOMElement
+ destroy(): void
+ findClosestDragDropTarget(e): void
+ indicateDropTarget(el): void
+ indicateDragTarget(el): void
+ dropTargetNodes(): Object
+ copyLongHTMLString(longStringActorPromise): void
MarkupContextMenu
-markup: Object
-inspector: Object
-selection: Object
-target: Object
-telemetry: Object
-toolbox: Object
-walker: Object
+destroy(): void
+show(event): void
-copyAttributeLink(link): void
-copyCssPath(): void
-copyImageDataUri(): void
-copyInnerHTML(): void
-copyOuterHTML(): void
-copyUniqueSelector(): void
-copyXPath(): void
-deleteNode(): void
-duplicateNode(): void
-editHTML(): void
-jumpToCustomElementDefinition(): void
-onAddAttribute(): void
-onCopyAttributeValue(): void
-onCopyLink(): void
-onEditAttribute(): void
-onFollowLink(): void
-onRemoveAttribute(): void
-pasteAdjacentHTML(position): Object
-pasteInnerHTML(): Object
-pasteOuterHTML(): Object
-showAccessibilityProperties(): void
-showDOMProperties(): void
-useInConsole(): void
-buildA11YMenuItem(menu): void
-getAttributeSubmenu(isEditableElement): Object
-getClipboardContentForPaste(): Object
-getDomBreakpointSubmenu(isElement): Object
-getNodeLinkMenuItems(): Array
-getPasteSubmenu(isElement, isFragment, isAnonymous): Object
-getPseudoClassSubmenu(isElement): Object
-openMenu(target, screenX, screenY): void
-updateA11YMenuItem(menuItem): void

Appendix B

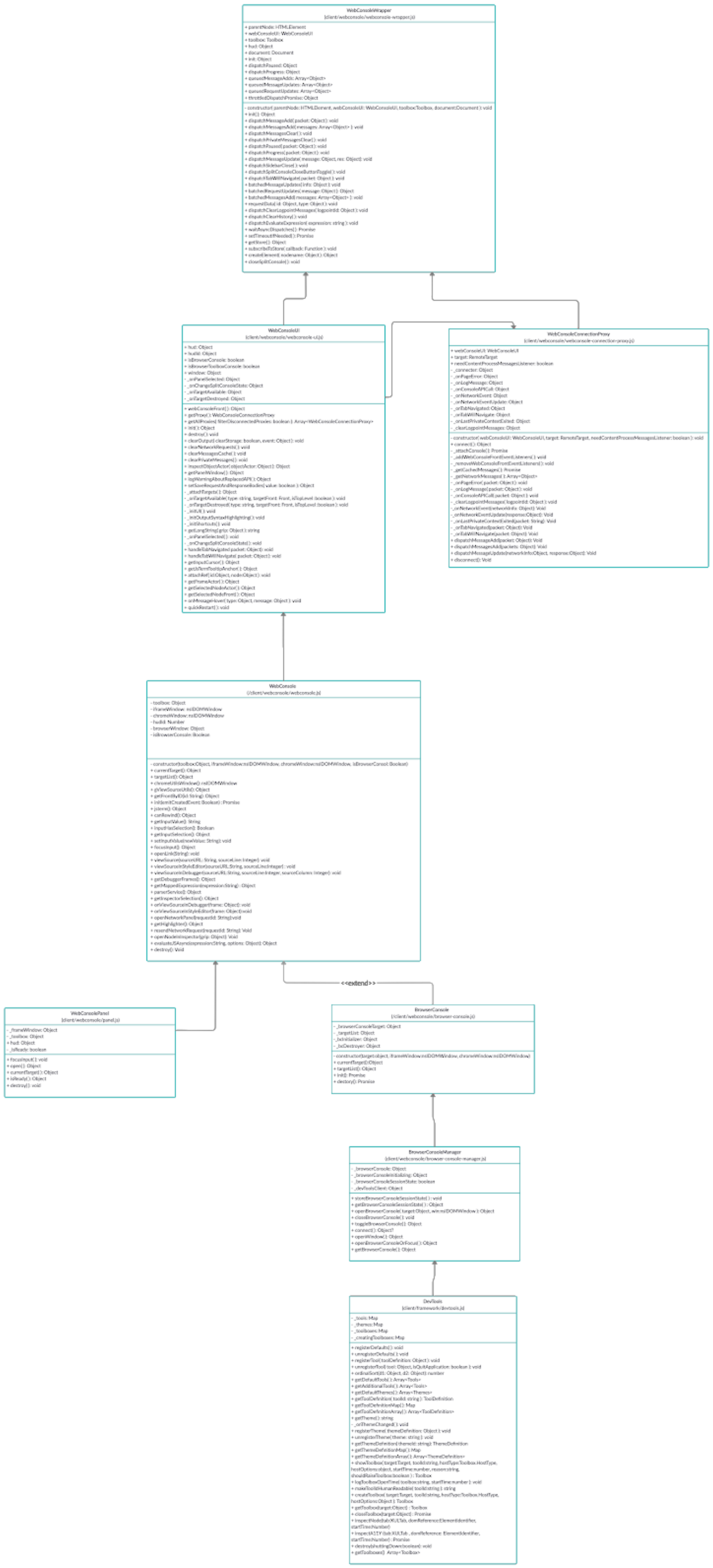

Root Actor
- actorID: int
- _parameters: var
- _onTabListChanged: var
- _onAddonListChanged: var
- _onWorkerListChanged: var
- _onServiceWorkerRegistrationListChanged: var
- _onProcessListChanged: var
- _extraActors: var
- _globalActorPool: var
+ applicationType: string
+ traits [boolean, boolean, boolean, boolean, boolean, var, boolean, int, boolean]
+ sayHello(): [int, string, string, [boolean, boolean, boolean, boolean, boolean, var, boolean, int, boolean]]
+ forwardingCancelled(string): [int, string, string]
+ destroy(): void
+ getRoot(): Object
+ listTab(string): void
+ getTab(string): Object
+ getWindow(int): Object
+ onTabListChanged(): void
+ listAddons(string): void
+ onAddonListChanged(): void
+ listWorkers(): void
+ onWorkerListChanged(): void
+ listServerWorkerRegistrations(): void
+ onServiceWorkerRegistrationListChanged(): void
+ listProcesses(): void
+ onProcessListChanged(): void
+ listRemoteFrames(int): void
<<abstract>>
BrowsingContextTargetActor
- BrowserTabList(Object): void
- DebuggerProgressListener(Object): void
- onWindowCreated(): void
- onWindowHidden(): void
- observe(): void
- onStateChange(): void
FrameTargetActorProxy
- _conn: Object
- _browser: string
- _form: Object
- _exited: boolean
- _options: Object
+ connect(): FrameTargetActor
+ update(Object): Object
+ form(): Object
+ exit(): void
WebExtensionActor
- deniedWarningDocuments: Object
+ listAddons(): WebExtensionActor
+ isSystemPrincipalWindow(Object): Object
+ createExceptionInfoResult(Object): Object
+ logAccessDeniedWarning(Object, Object, Object): Object
+ CustomizedReload(Object): Object
+ start(): void
+ observe(Object, Object, Object): void
+ onStateChange(Object, Object, Object, Object): void
+ stop(Object): void
FrameTargetActor
- Cr: var
- BrowsingContextTargetActor: var
- browsingContextTargetPrototype: var
- extend: var
- ActorClassWithSpec: var
- frameTargetSpec: var
- frameTargetPrototype: var
- _messageManager: Object
- _traits.reconfigure: boolean
- _sendForm: var
+ update(Object): Object
+ form(): Object
+ exit(): void
ChromeWindowTargetActor
- Ci: var
- BrowsingContextTargetActor: var
- browsingContextTargetPrototype: var
- extend: var
- ActorClassWithSpec: var
- chromeWindowTargetSpec: var
- chromeWindowTargetPrototype: var
+ getWindow(): ChromeWindowTargetActor
+ chromeWindowTargetPrototype._detach(): boolean
+ exports.ChromeWindowTargetActor(): void
+ chromeWindowTargetPrototype.observe(): void
chromeWindowTargetPrototype._attach(): boolean
WorkerTargetActor
- Ci: var
- ChromeUtils: var
- DevToolsServer: var
- XPCOMUtils: var
- protocol: var
- workerTargetSpec: var
- WorkerTargetActor: var
- detach(): void
- _preventServiceWorkerShutdown(): void
- _allowServiceWorkerShutdown(): void
+ listWorkers(): WorkerTargetActor
+ form(): void
+ attach(): void
+ detach(): void
+ destroy(): void
+ connect(Object): Object
+ push(): Object
+ onClose(): void
+ onError(): void
ParentProcessTargetActor
- Ci: var
- Services: var
- DevToolsServer: var
- getChildDocShells: var
- BrowsingContextTargetActor: var
- browsingContextTargetPrototype: var
- makeDebugger: var
- extend: var
- ActorClassWithSpec: var
- parentProcessTargetSpec: var
- parentProcessTargetPrototype: var
+ getProcess(): ParentProcessTargetActor
+ parentProcessTargetPrototype.observe(): void
+ parentProcessTargetPrototype._attach(): void
+ parentProcessTargetPrototype._detach(): void
ContentProcessTargetActor
- Cc: var
- Ci: var
- Cu: var
- ThreadActor: var
- Services: var
- WebConsoleActor: var
- makeDebugger: var
- Pool: var
- assert: var
- TabSources: var
- ActorClassWithSpec: var
- Actor: var
- contentProcessTargetSpec: var
_onWorkerListChanged(): void
+ getProcess(int): ContentProcessTargetActor
+ form(): Object
+ ensureWorkerList(): Object
+ listWorkers(): Object
+ pauseMatchingServiceWorkers(Object): Object
+ destroy(): void
WebExtensionTargetActor
- extend: var
- Ci: var
- Cu: var
- Cc: var
- Services: var
- ParentProcessTargetActor: var
- parentProcessTargetPrototype: var
- makeDebugger: var
- ActorClassWithSpec: var
- webExtensionTargetSpec: var
- DevToolsUtils: var
- FALLBACK_DOC_URL: string
- webExtensionTargetPrototype: var
+ connect(): WebExtensionTargetActor
+ webExtensionTargetPrototype._searchFallbackWindow:() Object
+ webExtensionTargetPrototype._destroyFallbackWindow(): void
+ webExtensionTargetPrototype._searchForExtensionWindow: Object
+ webExtensionTargetPrototype._onDocShellDestroy(Object): void
+ webExtensionTargetPrototype._onNewExtensionWindow(): void
+ webExtensionTargetPrototype._attach(): void
+ webExtensionTargetPrototype._detach(): void
+ webExtensionTargetPrototype._docShellToWindow(): void
+ webExtensionTargetPrototype._docShellsToWindows(): void
+ webExtensionTargetPrototype.isExtensionWindow(): boolean
+ webExtensionTargetPrototype.isExtensionWindowDescendent(): Object
+ webExtensionTargetPrototype._allowSource(): boolean
+ webExtensionTargetPrototype._onParentExit(): void

The above UML showcases the hierarchical actor structure found on the server-side of the application. The root actor functions as the genesis which gives birth to various other actors. When a parent actor is removed, so are its children. Actors are used for memory management

regarding various front-end processes throughout the application. For example, “Target actors” represent objects being targeted by a toolbox such as tabs, frames and more. These actors then give rise to children for more specific processes as well. The above are a few examples of many possible actors and their
relationships within the application.

Appendix C

The UML to the right showcases the internal architecture of the Console panel. The gist of it was discussed in detail above in The Console paragraph,


WebConsoleWrapper
(client/webconsole/webconsole-wrapper.js)
WebConsoleUI
(client/webconsole/webconsole-ui.js)
WebConsoleConnectionProxy
(client/webconsole/webconsole-connection-proxy.js)
WebConsole
(/client/webconsole/webconsole.js)
<<extend>>
WebConsolePanel
(client/webconsole/panel.js)
BrowserConsole
(/client/webconsole/browser-console.js)
BrowserConsoleManager
(client/webconsole/browser-console-manager.js)
DevTools
(client/framework/devtools.js)